

Automated Validation of Service-Oriented and API-Based Enterprise Systems for Reliable Digital Transformation in Saudi Arabia

Hossam Kenawy

0009-0006-1695-8282

Published October 2026

Abstract

The extent to which Saudi Arabia is undergoing a digital transformation now relies on enterprise platforms that are put together using APIs, service-oriented components, microservices, identity services, data stores, and capabilities that are managed externally. This architectural modularity speeds up the delivery process but at the same time spreads out any potential failures across the interface contracts, the parameter dependencies, the workflow state, the data semantics, the performance, the security, and the runtime operations. A service may therefore pass local tests yet the entire public or enterprise transaction end-to-end will still be unreliable. This review brings together recent evidence on automated validation methods that have been published between 2020 and 2025 and presents an integrated assurance framework aimed at achieving reliable digital transformation in Saudi Arabia. The review looks at specification-driven API testing, dependency-aware test generation, contract testing, cross-service functional and robustness testing, performance verification, security testing, and observability. Thirty peer-reviewed sources, verified by DOI, were chosen as part of a protocol-led evidence review with a focus on automation, architectural relevance, empirical support, and suitability for use in enterprise systems. The synthesis indicates that no one test technique offers enough assurance for distributed digital services; reliability instead comes from a layered body of evidence that links interface correctness with stateful workflows, service interoperability, resilience in the face of failure, security controls, and production telemetry. The paper suggests a seven-stage automated validation architecture and a continuous-assurance maturity pathway which transform diverse evidence into clear decisions regarding releases and governance. For Saudi organisations, the framework stresses interoperability, service continuity, auditable changes, and evidence-based risk control, thus connecting software engineering practice with the level of operational reliability required by national digital transformation programmes.

1 Introduction

The digital transformation taking place in Saudi Arabia has gone beyond the mere digitisation of individual administrative procedures to include the development of ecosystems in which government agencies, industries that are regulated, businesses, and technology providers share data and make use of each other's capabilities via software interfaces. APIs have become the main means of carrying out this kind of exchange since they allow services to be combined, reused, and made accessible across different organisations. Recent research focused on Saudi Arabia has connected the adoption of APIs with wider digital economic development and highlights the importance of interoperability, automation, innovation, and the ability to deliver services on a large scale [28]. However, the same level of connectivity that makes API-based transformation so attractive also gives rise to a reliability issue: a single enterprise transaction can pass through an API gateway, an identity provider, a number

of application services, shared data platforms, and external providers before a citizen or an employee is given the result. The fact that a process is correct at one point does not mean that the entire transaction is correct.

Because the service-oriented and microservice approaches increase the difficulty of the problem by dispersing the logic and responsibility among various components, services are able to evolve on their own, to use different technology stacks, and to be deployed at different times. As a result, testing research now views a system not as a single, monolithic program with a single point of execution but rather as a network of contracts and interactions. Early research into the testing of microservices highlighted the range of testing levels and the difficulty of reproducing faults in a distributed environment [1]. More recent studies have shown that automated testing can generate both valid and invalid requests from service specifications, follow stateful sequences of operations, observe the internal service chains, and deduce causal relationships between failures [2,8,10,11,21]. The importance of these developments lies in the fact that they move validation away from manually written checks of endpoints and towards the systematic generation of evidence throughout the architecture.

Yet the evidence is still fragmentary. While specification-based API testing is able to identify malformed responses and unexpected status codes it might fail to detect business-state corruption. Although contract testing can avoid syntactic incompatibilities between consumers and providers it does not pick up on semantic disagreements. Performance tests can confirm that service-level objectives are met when a representative workload is used but they cannot prove authorization is correct. Runtime observability can highlight abnormal latency, cascading failures, or security-related behaviour after a deployment has taken place, but it remains reactive unless telemetry is taken into account when making release decisions. Security testing introduces a further dimension since distributed APIs increase the attack surface and business-logic misuse may not be visible to conventional network controls [3,22]. Reliable digital transformation thus demands a validation strategy that combines these different kinds of evidence rather than viewing each one as a separate quality practice.

This requirement is particularly pertinent to the modernisation of Saudi Arabia's digital government and enterprise sectors. Research into the implementation of digital government in the Kingdom identifies organisational, technological, regulatory, and trust-related obstacles which have to be overcome if the digital services are to be sustainable [29,30]. Moreover, cybersecurity practices also have an impact on both the perceived and actual quality of e-government services [29]. It follows that reliability is not just something that concerns the testing team; it is an architectural and governance feature involving control over releases, the ability to trace evidence, operational accountability, and the capacity to detect degradation following deployment. The main question that this review sets out to answer is therefore: how can automated validation techniques be arranged into a coherent framework that provides defensible evidence of reliability for service-oriented and API-based enterprise systems in Saudi Arabia?

The work is based on conceptual ideas and draws on the more recent empirical software engineering literature. Rather than suggesting yet another separate testing algorithm, the review links the various techniques that function at different stages and points throughout the software lifecycle. It identifies evidence relating to the interface, dependencies, workflow, integration, resilience, security, performance, and runtime; determines in which cases existing methods are complementary or incomplete; and then turns this synthesis into an enterprise validation architecture. The framework that results is designed for situations in which failure can cross service boundaries and where

releases have to be supported by measurable evidence instead of just by the completion of a standard test checklist.

2 Review aim and objectives

This study aims to combine the current evidence regarding the automated validation of service-oriented and API-based enterprise systems and to develop an assurance framework suitable for the Saudi context which connects pre-release testing with operational reliability. The focus does not merely lie in defect detection; validation is seen as the generation of evidence to show that a distributed transaction remains acceptable even when the interfaces change, when data is transferred between services, when dependencies fail, when workloads vary, when security controls are put into effect, and when the behaviour in production differs from that in laboratory conditions.

The review is based on five objectives. The first of these is to identify the common areas of failure in respect of API contracts, parameter dependencies, stateful workflows, service chains, data exchange, performance, resilience and security. The second objective is to compare the various automated validation techniques according to the kind of evidence they use, the oracle or decision mechanism they employ, the architectural scope that they cover and the stage of the lifecycle at which they function. Thirdly, the review highlights the requirements that are most important to Saudi digital enterprises, namely interoperability, availability, auditable change, cybersecurity and cross-organizational trust. Fourthly, it combines the evidence into a layered validation architecture which makes the hand-offs between development, testing, deployment and operations clear. Fifthly, it establishes a maturity pathway along which organisations can progress from testing at isolated endpoints to achieving continuous, risk-based assurance without having to replace entirely the engineering tools currently in use.

3 Review methodology

A protocol-driven integrative review was employed since the research question involves various subfields of software engineering, each of which uses different empirical methods. The evidence base covers API test generation, service compatibility, microservice robustness, performance verification, security testing, observability, and studies concerning digital government in Saudi Arabia. The searches were based on combinations of terms such as web API, REST API, service-oriented architecture, microservices, contract testing, interoperability, automated validation, test generation, resilience, performance, observability, security, Saudi Arabia, and digital government. The sources were obtained from major computing and multidisciplinary indexes, such as Scopus, Web of Science, IEEE Xplore, the ACM Digital Library, and ScienceDirect. The publication period was limited to 2020–2025 in order to focus on current architectures and automation practices.

Eligibility was based on four criteria: the source had to have been peer reviewed, it had to be published in English within the specified time period, it had to be directly related to automated validation or to the Saudi digital-transformation context, and it had to be linked to a verifiable DOI. Studies were excluded if they concerned themselves only with the implementation technology and not with validation, if they described legacy service architectures that had only limited applicability to current API ecosystems, or if the evidence available was too weak to support the analytical aspect for which the study was intended. The final set of sources was deliberately restricted to thirty that were of high relevance so that each one could be looked at in terms of its method, evidence, and limitations rather than just being counted.

The data extraction followed a standard analytical framework. In the case of technical studies, the review noted the system type, the target of validation, the input evidence, the automation mechanism, the test oracle, the scope of execution, the empirical setting, and the reported limitations. For the studies set in the Saudi context, the extraction concentrated on the reliability-related implications such as interoperability, governance, cybersecurity, service quality, implementation barriers, and the strategic role of API-enabled infrastructure. The technical evidence was then assigned to six assurance areas: interface and contract correctness; parameter and state dependencies; cross-service integration and data consistency; functional robustness and resilience; security and resistance to misuse; and runtime observability. Performance was examined within the integration and resilience areas since capacity, latency, and resource behaviour usually only become apparent during multi-service execution.

The reason why comparison was used instead of statistical aggregation was that the various studies involve different kinds of outcomes, systems, and experimental designs. The evidence was analysed in relation to three questions: what kind of failure the method can detect, what part of the architecture it can observe, and what decision the resulting evidence can support? This prevents a typical category mistake in testing research, namely evaluating techniques only by the number of faults when these techniques function at different levels. A contract test which blocks an incompatible provider release and a resilience experiment which reveals cascading latency are both useful, but they address different reliability issues. The framework was therefore arrived at by combining complementary techniques into a lifecycle sequence and by determining the evidence that should carry over between the stages.

The two reference papers provided the structural basis for this synthesis without restricting the citation window. The first paper shows an example of an architecture based on components, with clearly defined web-service interfaces, modular business logic, microservice deployment, and a requirement for extensive interaction testing; the second arranges the distributed API interactions as observable structures and places emphasis on scalability, realistic validation, low-latency analysis, and robustness in dynamic conditions. These architectural principles are in line with the present review's focus on interfaces, integration paths, operational evidence, and explicit validation gates, just as the paper's main claims are supported by the 2020–2025 evidence corpus.

4 Evidence synthesis: Where automated validation must operate

The initial assurance boundary is the API contract; automated black-box testing becomes feasible thanks to OpenAPI and other interface descriptions since it is possible to generate requests without needing to have access to the internal workings of the service. Constraint-based generation can produce combinations based on the parameters as they are documented and can identify any discrepancies between an implementation and the interface it is supposed to have [2]. RESTest and similar methods highlight the benefits of specification-driven generation, whereas EvoMaster presents both black-box and white-box search strategies for REST APIs [4,5,8]. RestTestGen goes further by separating nominal scenarios from error scenarios and by making use of different oracles in order to detect abnormal results [10]. Together, these methods prove that interface specifications should be treated as executable validation assets rather than as passive documentation.

Complying with specifications is not enough since real APIs involve dependencies that are not captured by simple data types; parameters can be conditionally required, mutually exclusive, or valid only when another value is present. Research into inter-parameter dependencies has shown that

such rules are common and can be formally specified and analysed [9,27]. There is also a related issue arising from stateful behaviour: an operation may only be legal if another operation has previously created, authenticated, reserved, or modified a resource. Approaches based on modelling, such as Morest, make use of execution feedback in order to explore sequences of operations and improve the coverage of stateful behaviour [11,12]. Online testing also realizes that services similar to those in production can change in ways that make static pre-release assumptions obsolete [13]. For enterprise systems, the result is that validation must cover not just the endpoints but also the legitimate transitions between business states.

The next stage is cross-service interoperability. Even if a provider's local tests are passed, independent deployment and polyglot implementation can still cause problems for the consumers. In order to overcome this, consumer-driven contract testing is used, which expresses the expectations from the point of view of the consumer and checks these expectations against the provider before incompatible changes are released [26]. The benefit of this approach is limited but strategically significant since it turns compatibility from an informal coordination issue into an automatable checkpoint. However, syntactic compatibility should not be considered equivalent to end-to-end correctness; a request can still be structurally valid even if the meaning of the fields has changed, the units are interpreted differently, or an upstream service results in a logically inconsistent state. Cross-service validation must therefore integrate contracts with workflow assertions and data invariants.

The scope of functional and robustness testing for microservice architectures has moved beyond individual endpoints to include the entire execution chain. Recent research has involved producing both valid and invalid inputs based on service descriptions, carrying out these inputs through distributed architectures, monitoring both the external and internal services, and detecting the causal relationships between the failures that are observed [21]. This approach is significant since a failure that is visible at the system boundary may have its roots several hops earlier. It also aids in fault localization and stops development teams from attributing every failed transaction to the last service in the chain. The shift from testing endpoints to testing with an awareness of the path is a reflection of the architecture itself: reliability is an emergent property of the interactions, not the simple sum of the independently tested components.

Non-functional behaviour provides a second dimension of evidence. It is possible for microservices to be functionally correct yet to fail when subjected to realistic loads, deployment arrangements, or resource contention. By linking the carrying out of tests with reliability reasoning [14], integrated performance and reliability testing offers a solution. Another approach is through automated learning and verification of performance models: realistic workloads can be used to learn how a system behaves under different deployments and to quickly verify performance requirements [15]. Studies involving service meshes show that traffic-management policies themselves have an effect on latency and on operational behaviour [16]. These results justify a testing strategy according to which performance is assessed in relation to architectural configurations and failure scenarios rather than being tested once at the end with a standalone load test.

Security validation should be built in rather than being added as an afterthought. Research concerning cloud-service APIs has shown that the security properties of such APIs can be verified by systematically interacting with their REST interfaces [3]. The issue of excessive data exposure illustrates why response correctness cannot be judged simply by checking that the response conforms to the schema: an API might return a structure that is valid but still reveals more information than the caller is allowed to have [22]. The security metrics for microservices also highlight secure communic-

ation, identity management, and observability as measurable architectural issues [20]. A reliable validation programme must therefore not only meet the functional requirements but also test authorization boundaries, data minimization, identity propagation, and abnormal request behaviour. This is especially important in digital-government systems since there is a close link between service quality and cybersecurity [29].

In the end, runtime observability goes beyond validation at the stage of deployment. To explain the behaviour of short-lived components, cloud-native applications need coordinated logs, metrics, traces, and contextual metadata [18]. Research into adaptive observability shows that the way in which evidence is collected can be modified in accordance with the level of uncertainty and the forensic requirements [19]. This is important since pre-release environments are not able to replicate all the production dependencies, traffic patterns, or operational events. When telemetry is linked to specific service level objectives, release hypotheses, and incident criteria, it becomes part of the validation process. Hence, the best architecture establishes a closed loop in that runtime evidence should be used to inform new tests, specification repair, risk recalibration, and future release gates rather than staying confined to the operations dashboards.

Table 1. Evidence domains for automated validation of distributed enterprise systems.

Assurance domain	Primary failure surface	Automated methods	Evidence / oracle	Typical stage
Interface and contract	Schema drift, invalid response, undocumented behavior	Specification-based generation; consumer contracts	OpenAPI/schema; consumer expectation; response oracle	Build / pre-merge
Parameter and state	Illegal combinations, missing preconditions, broken state transition	Constraint solving; dependency models; sequence generation	Dependency rules; state model; business invariant	Component / integration
Integration and data	Cross-service mismatch, semantic corruption, inconsistent identifiers	Path-aware execution; end-to-end assertions; reconciliation checks	Service chain; data invariants; transaction trace	Integration / staging
Resilience and performance	Cascading latency, overload, failed dependency, poor recovery	Load generation; fault injection; model verification	SLOs; latency; error budget; recovery behavior	Staging / pre-release
Security	Authorization failure, excessive exposure, identity propagation defect	Negative security tests; misuse cases; fuzzing	Policy; identity context; data-minimization oracle	All pre-release stages
Runtime assurance	Unknown production path, silent degradation, incomplete incident evidence	Adaptive logging; metrics; distributed tracing	Telemetry; SLO breach; incident evidence	Production / continuous



Figure 1. Layered automated validation architecture linking specifications, cross-service evidence, operational testing, runtime observation and release governance.

5 Proposed automated validation framework

Figure 1 presents the literature in the form of a seven-stage architecture. At the first stage a machine-readable evidence base is created from the API descriptions, schemas, the expectations of the customers, the identity policies, the business rules, and the known service dependencies. The aim here is not to centralize all the artifacts but to ensure that there is a traceable link between a reliability claim and the evidence used to test it. In large and complex enterprises, undocumented assumptions are a major cause of failure. While a declared contract, dependency rule, or service-level objective can be verified automatically, an assumption that exists only in a developer's memory cannot.

In the second stage, contracts and interfaces are verified. Before the integration environments become costly, schema validation, response-code oracles, boundary-value generation, negative tests, dependency constraints, and consumer expectations are carried out. Search-based and specification-based generators are able to expand the input space while at the same time maintaining repeatability [4,5,7,8,10]. Although this stage should reject any incompatible changes early on, the fact that it passes means only that the interfaces are internally and syntactically acceptable; the evidence must still be tied to the exact service version so that any future failures can be traced back to a known contract baseline.

In the third stage, the state and the workflow are verified. In this case, the test model shows sequences instead of individual calls: authentication can come before authorization; resource creation can come before modification; and an approval can come before payment or notification. Model-based and feedback-directed generation are useful since they systematically examine both legal and illegal transitions [11,12]. The inter-parameter constraints are incorporated at this stage so that the state is not tested using inputs that are semantically impossible [9,27]. The main oracle consists of a set of business invariants, namely state transitions, idempotency, authorization continuity, and permitted side effects.

At the fourth stage, integration and the consistency of data across service boundaries are assessed. Contract tests are used to check whether the consumer and provider are compatible, and path-aware functional tests monitor the internal service chain [21,26]. Data assertions must compare identifiers, units, reference integrity, temporal ordering, and the results of reconciliation at key boundaries. It is also in this stage that external dependencies and legacy adapters should be represented explicitly. The enterprise architecture reference provided shows why this is necessary: since componentized business logic, web-service APIs, identity management, persistence, and external providers result in a number of independently replaceable components, their interactions still need to be validated end-to-end.

The fifth stage involves resilience, security, and performance. Instead of viewing these as separate specialist activities, the framework runs typical workloads while at the same time delaying, making certain dependencies unavailable, misconfiguring them, or exposing them to invalid and unauthorized requests. Quantitative evidence regarding response time and scalability is obtained as a result of the performance verification [14,15]. The security tests look at access control, exposure, and misuse [3,22]. The robustness tests check whether faults remain contained or spread throughout the service chains [21]. The aim is to find out if the architecture can maintain an acceptable level of service under adverse conditions, not just to see if it recovers from an artificial outage.

The sixth stage is runtime observability, involving the linking of release evidence to logs, traces, metrics, audit events, and service-level indicators so that the behaviour after a deployment can either confirm or refute the assumptions that were made before the release [18,19]. If a release has passed latency validation but ends up creating a new cross-service bottleneck in production, it should lead to specific regression and capacity tests. Similarly, an authorization anomaly should be used to enhance the negative test suites. In this way, production telemetry becomes an input for the engineering process rather than remaining in a separate operational archive.

The seventh stage involves an evidence-based release and governance checkpoint. The decision is not based on a single percentage score but rather on a structured assessment of whether the necessary claims have adequate evidence to support them: that the interfaces remain compatible; that the critical workflows work successfully; that the data invariants are maintained; that the failure modes are contained; that the security controls function as intended; that performance stays within the organisation's service objectives; and that observability is able to detect any relevant degradation. In cases where evidence is missing, the gate notes down an explicit residual risk together with an accountable owner. This method of governance is more defensible than relying on a general pass rate since it maintains the link between business criticality and the level of validation.

6 Implications for Saudi digital transformation

In Saudi Arabia the layered approach is particularly valuable since the country's digital transformation relies on large ecosystems rather than on individual applications. Growth based on the use of APIs promotes data sharing, platform integration and the development of new digital services [28]. Moreover, research into the implementation process points to various obstacles relating to organizational readiness, regulation, infrastructure, coordination and user trust [30]. While automated validation cannot overcome all of these problems, it can lessen a particularly costly source of uncertainty—that is, whether a modified digital service still interoperates reliably with the rest of the ecosystem.

With regard to government platforms, reliability must be understood as the assurance of a continuous outcome for individuals or businesses when moving across different institutions. A transaction

might involve identity, authorization, registry data, payment, notification, and services that are specific to a sector. The scope of validation should therefore be based on the transaction rather than on the organizational structure. It thus makes sense to keep a catalog of the key user journeys and to link each journey to contracts, state transitions, external dependencies, data invariants, service objectives, and security obligations. Automation can then decide which tests to carry out according to the extent of impact of a change. A change to a shared identity or registry interface should require more extensive evidence than a change to an isolated presentation component.

Interoperability is just as important. Saudi organisations will probably integrate modern cloud-native services with their existing enterprise systems, packaged software, and systems that are operated by external parties. Using a component-based architecture with clearly defined service interfaces makes it easier to replace and extend components, but it also introduces risks relating to versioning and compatibility. Practical interoperability controls can be achieved through the use of consumer-driven contracts, dependency-aware API tests, and end-to-end data assertions. The framework does not demand that all agencies or suppliers use the same implementation stack; instead, it requires them to express testable expectations and to keep traceable evidence of those expectations.

Cybersecurity must be regarded as a component of service quality rather than as part of a separate compliance programme. Research carried out in the Kingdom shows that cybersecurity practices have an effect on the quality of e-government services [29], whereas more general studies in Saudi Arabia highlight data-security management as an ongoing necessity for electronic government [24]. The framework suggested therefore includes identity, authorization, exposure, resistance to misuse, and auditability within the same validation process as functional correctness. This is especially significant in the case of APIs since a response can be technically successful but still breach the principle of least privilege or reveal unnecessary data. Security evidence should therefore be assessed at both the transaction and service-chain levels, covering the propagation of identity across internal calls.

Governance is also improved by having up-to-date validation evidence. Since large transformation programs include frequent releases, a number of suppliers, and constantly changing dependencies, a signed test report that has been produced before go-live is too static for this kind of environment. The evidence should be versioned together with service contracts, deployment artifacts, and release decisions. If a provider alters an interface, the relevant consumer contracts and business journeys can be automatically identified. When production telemetry shows an unexpected coupling, the architecture model and the regression suite can be updated. This level of traceability enables accountability without requiring all decisions to be made in a central testing organization.

In the end, the framework offers a way of adopting it gradually. Organizations that have only a limited degree of automation can start off by making the API specifications executable and adding contract tests to the more risky interfaces; the following stage involves stateful workflow and integration validation, then comes the automation of resilience, performance, and security. Observability then brings the entire process to a close and allows risk-based decisions regarding releases. This sequence is shown in Figure 2 and is intentionally carried out in stages since transformation programmes have to enhance assurance without at the same time stopping service delivery; a framework that demands a complete replacement of the tooling before it starts to deliver value is not likely to be taken up on an enterprise scale.

Continuous Assurance Maturity Pathway

Progression from endpoint checks to evidence-driven enterprise reliability

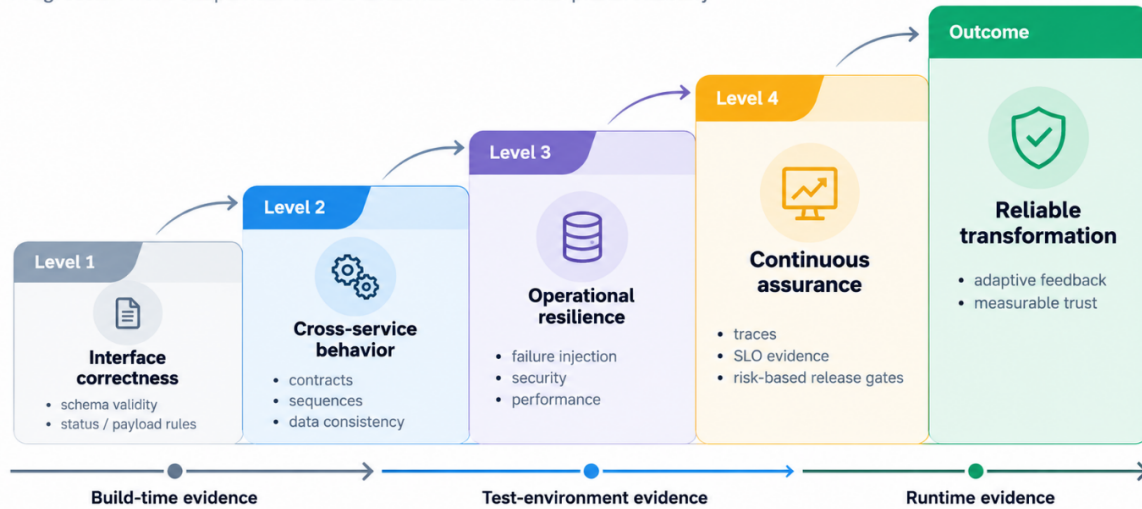


Figure 2. Continuous-assurance maturity pathway from interface correctness to evidence-driven reliability and adaptive operational feedback.

Table 2. Recommended governance matrix for validation evidence in Saudi digital enterprise systems.

Control domain	Reliability claim	Pre-release evidence	Runtime signal	Saudi enterprise rationale
API compatibility	Provider change cannot violate known consumer expectations	Contract pass linked to exact provider and consumer versions	Compatibility error rate; rejected incompatible build	Interoperability and supplier coordination
Critical journey	End-to-end transaction preserves required state and data	Journey-specific workflow and reconciliation evidence	Transaction success; invariant violations; retry/idempotency anomalies	Continuity of citizen and enterprise outcomes
Resilience	Critical service remains acceptable under dependency degradation	Fault scenario evidence plus recovery observation	Organization-defined SLO; recovery time; propagated failure count	Operational continuity and blast-radius control
Security and privacy	Identity, authorization and response exposure remain within policy	Negative authorization and exposure tests; audit evidence	Denied unauthorized actions; anomalous exposure; identity-propagation failures	Trust, data protection and accountable access
Performance	Representative workloads satisfy service objectives	Load evidence by deployment configuration	Latency percentiles; saturation; throughput; resource headroom	Scalable digital-service delivery
Observability	Operational evidence can detect and explain relevant degradation	Trace/metric/log coverage mapped to critical journeys	Unexplained error ratio; trace completeness; alert precision	Auditable operations and faster incident learning
Release governance	Residual risk is explicit and owned	Evidence bundle with exceptions, owner and expiry/review point	Open high-risk exception count; evidence freshness	Accountable change across multi-supplier environments

7 Research gaps and future agenda

The literature points out four areas that warrant in-depth research. Firstly, whilst test generation is more advanced at the interface level than at the semantic business level, it still requires human input when it comes to domain invariants. Although tools are able to deduce values, status expectations, and operation sequences, domain invariants generally need to be specified by people. Research should look into how enterprise policies, process models, and historical transaction evidence can be turned into machine-checkable oracles without making them opaque. The case of Saudi digital services offers a valuable setting for examining semantic validation because it involves bilingual data, national identifiers, sector-specific regulation, and cross-agency workflows.

Secondly, interoperability research should place greater emphasis on change propagation. While consumer-driven contracts safeguard established consumer expectations, enterprises also need to grasp the indirect dependencies and data transformations that occur several hops downstream. By combining graph-based architecture models, distributed tracing, and repository mining it is possible to determine which key journeys are likely to be impacted by a suggested interface change. This kind of impact analysis would enable test selection to be based on risk rather than relying on a generic regression suite.

Thirdly, in many studies the pre-release and runtime evidence are only weakly linked. Research into observability shows how to collect and adjust operational evidence [18,19], and research into testing explains how to produce controlled inputs. Future systems should link the two together: traces can be used to identify under-tested paths, the distributions of production parameters can be used to improve test data, incident signatures can act as regression oracles, and failures to meet service objectives can lead to targeted performance experiments. Thus, continuous delivery could be transformed into continuous validation rather than just involving more frequent deployments.

Fourthly, there are no standardised evidence models in regard to assurance governance. A high rate of tests passing may mask the fact that a vital dependency has not been covered, and raw coverage figures can lead to a preference for a large number of low-value tests. Research should aim to establish assurance cases for service-oriented systems that connect business criticality, architectural risk, validation evidence, operational telemetry, and residual risk. These models would be especially useful in digital-government environments which involve more than one supplier since they allow expectations to be verified without specifying a particular development method or toolchain.

An important fifth area for research is that of validation economics. Although automation is typically defended on the grounds of execution speed, the more significant question for an organisation is whether the evidence leads to a reduction in the expected cost of failure. Future research should involve modelling the relationship between the extent of validation, the criticality of a change, the number of defects that escape, the length of time incidents take to resolve, the amount of engineering effort required, and the level of service disruption. This would enable organisations to find out where advanced stateful generation, fault injection, or adaptive observability gives the highest additional value. In the case of a national digital ecosystem, the analysis must also take into account dependency concentration: since a single failure can impact a number of services at the same time, a shared platform can warrant a higher level of assurance. Prioritising in an evidence-based manner would make continuous assurance financially justifiable to programme leaders rather than merely technically appealing.

8 Limitations

The review has a number of limitations. The time frame within which the evidence is drawn gives more weight to current API and microservice practices and omits earlier fundamental work unless it has been included via more recent studies. The final set of sources was intentionally limited to thirty items that have been verified by DOI, a move which allowed for a more detailed comparative analysis but means that it does not cover every testing tool or industrial practice. There are more experiments recorded for publicly available or research systems than for high-criticality government systems, since in the latter case details about the data and the architecture are frequently restricted. As a result, the framework suggested should be seen as an architecture for validation based on available evidence and not as a quantitatively proven Saudi reference model.

Another limitation is the diversity of the outcome measures. Papers that assess APIs mention faults, deviations in status codes, coverage, and the tests generated; studies involving microservices may report service-chain failures, latency, resource consumption, and diagnostic accuracy; whilst governance studies make use of organisational and service-quality factors. It would therefore be misleading to attempt a direct numerical ranking. Instead, the synthesis concentrates on the kind of reliability claim that each method is able to support. Future empirical research should apply the entire framework to typical Saudi enterprise journeys and measure defect escape, the change-failure rate, recovery behaviour, security findings, release lead time, and operational service objectives over a number of release cycles.

9 Conclusion

It is not possible to guarantee a reliable digital transformation by testing each API on its own. Enterprise systems based on a service-oriented approach fail as a result of their interactions: a compatible endpoint may be involved in an invalid workflow; a correctly designed workflow can deteriorate when under load; an available service might expose too much data; and a release that works well in a test environment can act differently when the production dependencies and traffic patterns change. The evidence examined between 2020 and 2025 thus supports a multi-layered approach to automated validation in which contracts, dependencies, workflows, integration paths, resilience, security, performance, and runtime observability provide separate but complementary assurance.

For Saudi Arabia, this method brings software reliability in line with the real-world requirements of digital government and the modernisation of enterprises—specifically addressing the need for interoperability across different platforms, the continuous delivery of essential services, the secure sharing of data, the ability to audit changes, and the capacity to learn from operational experience. The suggested seven-stage architecture turns these requirements into an engineering process, while the maturity pathway provides a way of adopting the approach gradually. Its main idea is straightforward: confidence should be based on traceable evidence relating to the risks involved in a distributed transaction, not on the number of tests that have been carried out. If organisations keep this evidence throughout both development and operations, they will be able to make quicker changes without having to treat reliability as an activity carried out after the fact. In this way, automated validation becomes the foundation of trustworthy transformation rather than just a quality-control measure. This evidence-based approach also makes it possible to transfer assurance between suppliers, since a release can be examined on the basis of its claims, artifacts, and observed behaviour rather than on the basis of undocumented local practices. It prompts teams to regard specifications, tests, traces, and

service objectives as linked engineering assets whose value grows when they can be traced back to the same business process. Over time, this traceability will help to eliminate redundant testing, make clear who is responsible for failures that span multiple services, and enable architectural risks to be seen before incidents force organisations to rework their dependencies under pressure. The framework is therefore suitable for iterative delivery: the level of validation can be increased according to the level of risk while lower-impact changes are still able to move quickly through the automated checkpoints. As a result, future validation programmes should maintain transparent evidence at all interfaces.

References

- [1] Waseem M, Liang P, Márquez G, Di Salle A. Testing microservices architecture-based applications: a systematic mapping study. In: *27th Asia-Pacific Software Engineering Conference*; 2020. p. 119–128. <https://doi.org/10.1109/APSEC51365.2020.00020>
- [2] Martin-Lopez A, Segura S, Ruiz-Cortés A. RESTest: black-box constraint-based testing of RESTful web APIs. In: *Service-Oriented Computing*; 2020. p. 459–475. https://doi.org/10.1007/978-3-030-65310-1_33
- [3] Atlidakis V, Godefroid P, Polishchuk M. Checking security properties of cloud service REST APIs. In: *IEEE International Conference on Software Testing, Validation and Verification*; 2020. <https://doi.org/10.1109/ICST46399.2020.00046>
- [4] Arcuri A, Galeotti JP, Marculescu B, Zhang M. EvoMaster: a search-based system test generation tool. *Journal of Open Source Software*. 2021;6(57):2153. <https://doi.org/10.21105/joss.02153>
- [5] Arcuri A. Automated black- and white-box testing of RESTful APIs with EvoMaster. *IEEE Software*. 2021;38(3):72–78. <https://doi.org/10.1109/MS.2020.3013820>
- [6] Sigalov K, Ye X, König M, Hagedorn P, Blum F, Severin B, et al. Automated payment and contract management in the construction industry by integrating building information modeling and blockchain-based smart contracts. *Applied Sciences*. 2021;11:7653. <https://doi.org/10.3390/app11167653>
- [7] Zhang M, Arcuri A. Adaptive hypermutation for search-based system test generation: a study on REST APIs with EvoMaster. *ACM Transactions on Software Engineering and Methodology*. 2021;31(1). <https://doi.org/10.1145/3464940>
- [8] Martin-Lopez A, Segura S, Ruiz-Cortés A. RESTest: automated black-box testing of RESTful web APIs. In: *International Symposium on Software Testing and Analysis*; 2021. <https://doi.org/10.1145/3460319.3469082>
- [9] Martin-Lopez A, Segura S, Müller C, Ruiz-Cortés A. Specification and automated analysis of inter-parameter dependencies in web APIs. *IEEE Transactions on Services Computing*. 2022;15(4):2342–2355. <https://doi.org/10.1109/TSC.2021.3050610>
- [10] Corradini D, Zampieri A, Pasqua M, Viglianisi E, Dallago M, Ceccato M. Automated black-box testing of nominal and error scenarios in RESTful APIs. *Software Testing, Verification and Reliability*. 2022;32. <https://doi.org/10.1002/stvr.1808>
- [11] Liu Y, Li Y, Deng G, Liu Y, Wan R, Wu R, et al. Morest: model-based RESTful API testing with execution feedback. In: *International Conference on Software Engineering*; 2022. <https://doi.org/10.1145/3510003.3510133>
- [12] Liu Y, Li Y, Liu Y, Wan R, Wu R, Liu Q. Morest: industry practice of automatic RESTful API testing. 2022. <https://doi.org/10.1145/3551349.3559498>
- [13] Martin-Lopez A, Segura S, Ruiz-Cortés A. Online testing of RESTful APIs: promises and challenges. In: *ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*; 2022. <https://doi.org/10.1145/3540250.3549144>
- [14] Camilli M, Guerriero A, Janes A, Russo B, Russo S. Microservices integrated performance and reliability testing. In: *International Conference on Automation of Software Test*; 2022. p. 29–39. <https://doi.org/10.1145/3524481.3527233>
- [15] Camilli M, Janes AA, Russo B. Automated test-based learning and verification of performance models for microservices systems. *Journal of Systems and Software*. 2022;187:111225. <https://doi.org/10.1016/j.jss.2022.111225>

-
- [16] Sedghpour MRS, Klein C, Tordsson J. An empirical study of service mesh traffic management policies for microservices. In: *ACM/SPEC International Conference on Performance Engineering*; 2022. p. 17–27. <https://doi.org/10.1145/3489525.3511686>
- [17] Ehsan A, Abuhaliqa MAME, Catal C, Mishra D. RESTful API testing methodologies: rationale, challenges, and solution directions. *Applied Sciences*. 2022;12:4369. <https://doi.org/10.3390/app12094369>
- [18] Kosińska J, Baliś B, Konieczny M, Malawski M, Zieliński S. Toward the observability of cloud-native applications: the overview of the state-of-the-art. *IEEE Access*. 2023;11:73036–73052. <https://doi.org/10.1109/ACCESS.2023.3281860>
- [19] Monteiro D, Yu Y, Zisman A, Nuseibeh B. Adaptive observability for forensic-ready microservice systems. *IEEE Transactions on Services Computing*. 2023;16(5):3196–3209. <https://doi.org/10.1109/TSC.2023.3290474>
- [20] Zdun U, Quèval PJ, Simhandl G, Scandariato R, Chakravarty S, Jelic M, et al. Microservice security metrics for secure communication, identity management, and observability. *ACM Transactions on Software Engineering and Methodology*. 2023;32(1):1–34. <https://doi.org/10.1145/3532183>
- [21] Giamattei L, Guerriero A, Pietrantuono R, Russo S. Automated functional and robustness testing of microservice architectures. *Journal of Systems and Software*. 2024;207:111857. <https://doi.org/10.1016/j.jss.2023.111857>
- [22] Pan L, Cohnsey S, Murray T, Pham VT. EDEFuzz: a web API fuzzer for excessive data exposures. In: *International Conference on Software Engineering*; 2024. <https://doi.org/10.1145/3597503.3608133>
- [23] Golmohammadi A, Zhang M, Arcuri A. Testing RESTful APIs: a survey. *ACM Transactions on Software Engineering and Methodology*. 2024;33(1). <https://doi.org/10.1145/3617175>
- [24] Alharbi AS, Halikias G, Rajarajan M, Yamin M. A review of effectiveness of Saudi E-government data security management. *International Journal of Information Technology*. 2021;13:573–579. <https://doi.org/10.1007/s41870-021-00611-3>
- [25] Hui M, Wang L, Li H, Yang R, Song Y, Zhuang H, et al. Unveiling the microservices testing methods, challenges, solutions, and solution gaps: a systematic mapping study. *Journal of Systems and Software*. 2025;220:112232. <https://doi.org/10.1016/j.jss.2024.112232>
- [26] Schwarz GD, Quast F, Riehle D. Ensuring syntactic interoperability using consumer-driven contract testing. *Software Testing, Verification and Reliability*. 2025;35(5):e70006. <https://doi.org/10.1002/stvr.70006>
- [27] Barakat S, Martin-Lopez A, Müller C, Segura S, Ruiz-Cortés A. The IDL tool suite: specifying and analyzing inter-parameter dependencies in web APIs. *SoftwareX*. 2025;29:101998. <https://doi.org/10.1016/j.softx.2024.101998>
- [28] Ali MA, Salih SM. Impact of application programming interfaces (APIs) economy on digital economics in Saudi Arabia. *Sustainability*. 2025;17:4104. <https://doi.org/10.3390/su17094104>
- [29] Al-Hawamleh AM. Investigating the multifaceted dynamics of cybersecurity practices and their impact on the quality of e-government services: evidence from the KSA. *Digital Policy, Regulation and Governance*. 2024;26(3):317–336. <https://doi.org/10.1108/DPRG-11-2023-0168>
- [30] Makki AA, Alqahtani AY. Modeling the barriers surrounding digital government implementation: revealing prospect opportunities in Saudi Arabia. *Sustainability*. 2022;14:15780. <https://doi.org/10.3390/su142315780>