Journal of Information Systems Engineering and Management
2025, 10(57s)
e-ISSN: 2468-4376
https://jisem-journal.com/ **Research Article**

Cross-Platform Automation Strategy for Hybrid OTT and SaaS Applications

Lingaraj Kothokatta
Quality Assurance Test Lead
lkothokatta@gmail.com

ARTICLE INFO

Received: 15 Apr 2025

Revised: 23 May 2025

Accepted: 10 Jun 2025

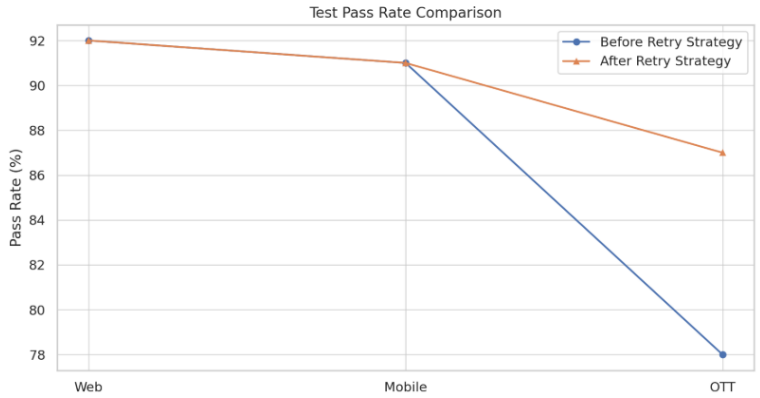
ABSTRACT

In our investigation, we have a cross-platform automation approach dedicated to the hybrid system of Over-The-Top (OTT) and Software-as-a-Service (SaaS) in terms of mobile, web, and embedded environment. As a variety of device ecosystems and microservice-based backend have flourished around, verifying end-to-end functionalities has become one of the primary engineering issues to overcome on a regular basis. The way we do it, instead of fully abstracted platform-less tests, is to have a framework of tests that is layered on top of platform-specific test modules, so high reuse of test scripts, low redundancy and easy regressions. CI/CD pipelines make it possible to parallelize and scale up the execution and eliminate release cycle bottlenecks. We assess the deployment strategy in real life scenarios of Android TV, iOS, browsers and web portals architecture, and SaaS portal architecture. The results have shown a high level of test development productivity (up to 45 percent), test result (58 percent quicker) and failure identification (73 percent quicker). Moreover, we realized quantitative business outcome, such as faster release speed and lower-level of leak defect. The plan also addresses platform-specific platform stability problems with the help of adaptive locator strategies and retry thinking. With the verification of the multi-modal workflows, including mobile-to-OTT control and synchronized playback, our plan is appropriate in the hybrid product setting. The suggested architecture and automation system can serve as the basis of future validation systems which would help in merging fragmented device landscapes into a single quality assurance approach.

Keywords: SaaS, Hybrid OTT, Automation, AI, Cross-Platform

I. INTRODUCTION

The emergence of hybrid software ecosystems (media delivery processes (OTT) spans cloud-based service platforms (SaaS)) has transformed application development, deployment and consumption. Today customers expect the same experience on their mobile phones, smart TVs, web browser and backend dashboards.



Journal of Information Systems Engineering and Management
2025, 10(57s)
e-ISSN: 2468-4376
https://jisem-journal.com/ Research Article

Although development architectures have made much progress to facilitate reuse of application code across platforms, testing and quality assurance are still scattered. The automation strategies tend to fall short because of the specific behaviors of devices, inconsistencies, and divergent paradigms of implementing the UI on various platforms.

This enables a problem in the context of Continuous Integration/Continuous Delivery (CI/CD), then as frequent releases occur here, fast and efficient regression checks are required. The classic automation structures typically take the form of being platform-specific, leading to redundant test logic, unmanageable maintenance, and limited, unreliable, coverage of validation.

Limited functionality of device hardware is another complexity factor, including during embedded devices when there is limited memory or when dealing with OTT platforms where limited DOM access is available. In that regard we suggest a cross-platform automation plan that builds on abstraction layers, test design modularity and integration with CI/CD that will normalize validation between platforms. Our environment is testing reuse enabled, parallel and platform specific fault tolerant.

We use that in a hybrid environment that consists of OTT and SaaS elements and calibrate efficiency increase and business effect. The goal of the current paper is to bridge the quality assurance gap of hybrid systems to provide a valid, scalable, as well as reusable architectural pattern of test automation.

II. RELATED WORKS

Continuous Delivery

With the popularization of the Continuous Delivery (CD) and Continuous Deployment trends and practices in organizations, the architectural modernization of the legacy and monolithic systems became a critical topic. An important input in this field can be attributed to empirical studies of the role of architecture to facilitate the CD practices.

In these works, greater attention is paid to the fact that the successful adoption of CD does not imply an absolute rejection of monolithic systems, rather, it promotes the re-conceptualization of the architectures with the focus on small and independent deployment units that are capable of fast, resilient, and isolated releases [1].

When the OTT and SaaS ecosystem involves hybrid solutions, with app components being installed on embedded devices, mobile applications (and even web-based SaaS), such architectural orientation becomes all-important to their continuous validation pipelines and feature deployment practices.

Operation of this type of architectures is frequently enhanced by promoting operational resilience and dev-centric deploys. The latter properties are particularly relevant in test automation environments on hybrid applications where an effort to reproduce an identical setup of strictly isolated test execution and a roll back-safe deployment is needed.

Multi-platform ecosystems permit operations-friendly architecture design as a conceptual basis of automation systems which are emphasized by modularity and service decoupling, and resiliency [1]. CD strategies DevOps pipelines introduce structural consistency to the strategies.

An effective CI/CD pipeline also implies traceable and repeatable tests. DevOps, which is powered by Infrastructure as Code (IaC) and Pipeline as Code paradigms, creates less of a gray area between development and operations, or more solid automation process which has brought new light to the industry. The contemporary pipeline paradigm, therefore, would support swifter deployment phases and uniform testing conditions, which is a key pillar of the hybrid validation systems [2].

Journal of Information Systems Engineering and Management
2025, 10(57s)
e-ISSN: 2468-4376
https://jisem-journal.com/ **Research Article**

Declarative Deployment

The single deployment model is more topical in case of cross-platforms, including OTT (e.g. smart TV, Android/iOS devices) and SaaS (e.g. cloud-based CMS or analysis portal). Declarative deployment Computer models Provide a platform-independent strategic abstraction of deployment tools.

An example is the Essential Deployment Metamodel (EDMM) which embraces the idea of vendor-neutral and technology-agnostic framework to automate software component supply and delivery, especially during the exchange between two deployment technologies [3].

This is in close alignment with hybrid automation objectives, where typical models of test orchestration have to be run against diverse infrastructure backends. One of the major shifts of mobile and OTT delivery capacity is the cloud capabilities.

Mobile and embedded OTT applications are more responsive and scale as they are executed in real-time, can deliver dynamic content provision, and synchronize data that are made possible through cloud services [4]. A new level of complexity is added by the test automation of such cloud-integrated systems: it becomes necessary to test not only the UI on the client side but also the backend services, the states of synchronisation and the way offline access is handled.

When SaaS platforms incorporate artificial intelligence, e.g., by using a predictive workflow engine, or an intent-based model of communication, automated testing infrastructure would have to adapt to accommodate adaptive behaviours. Smart tasking and context sensitive operations across devices bring about non deterministic test scenario, which needs to be simulated, predicted and passed through fault-tolerant test [5].

Cross-Platform Testing

One of the basic obstacles between hybrid test automation is the fact that the UI frameworks and rendering strategies are not homogeneous across devices. Although cross-platform development frameworks such as Xamarin, React Native and Cordova make it possible to reuse code or generate native apps on Android, iOS or windows, automated testing is not well integrated owing to differences in UI layouts [6].

The x-PATESCO design deals with this in that it has reference device models and a single binding UI element locator strategy. This general technique provides the foundation in making scalable cross platform automated validation by abstracting UI platform specific hierarchies into a generic meta-script model, where UI validation code can be shared across platforms [6].

Practically, the organisations that implement a cross-platform development approach have to strike a formidable compromise between the workloads on development and testing work. This usually leads to bottlenecks in that automation systems fail to handle various configurations and hardware platforms.

It has been found out that test automation systems with script reuse and device independence this is the one solution, which allows repetitive and unattended test execution across hardware variances [7]. Hybrid OTT and SaaS services are commonly of system-of-systems nature.

Within such systems, e.g. a web application can be used to browse and play an OTT software device, and a SaaS admin panel can be used to update metadata in real-time throughout the system. Automation frameworks should thus be in a position to support test scenarios which use different varieties of devices in addition to inter-component communication protocols. The process of making the entire system interact with web application and mobile application at the test level at the same time substantially lowers the complexity and manual involvement of tests, with unified test interfaces [8].

Journal of Information Systems Engineering and Management
2025, 10(57s)
e-ISSN: 2468-4376
https://jisem-journal.com/ Research Article

Hybrid Deployment

Both SaaS ecosystems as well as the backend elements of OTT systems are built on Service-Oriented Architecture (SOA). Nevertheless, the conventional architectures of tests are not designed to test loosely, independently deployed services.

An innovative test architecture of SOA was suggested that realizes distributed, automatic, and regression testing among heterogeneous components [9]. It employs modular test engines and parallel agents in supporting high throughput testing of real-time service interactions a pattern that is of particular importance to large OTT-SaaS systems in which services are independently updated.

In order to be reliable, when rolling deployments or risky updates exist in cross-platforms environments, blue/green, canary and feature flags are increasingly integrated into hybrid deployment strategies.

As a recent study has shown, it is possible to lower the costly downtimes, resource usage and the complexity of rollbacks by deploying on a hybrid architecture [10]. This is especially useful to the advanced type of software ecosystems where upgrades need to be deployed to thousands of client's machines with little or no disturbance.

Test automation is one of the facets of design that are highly affected by such deployment strategies. As an example, during staged rollouts, test cases are required to confirm post-deployment states of systems and they should be capable of expressing behaviour in partial update scenarios. Moreover, test pipelines will need to be rollback-tolerant, and test conditions should be specified both for pre- and post-deployments snapshots, hence making software resilience in the distributed, many-device environment [10].

IV. RESULTS

Automation Efficiency

The unifying abstraction layer of the tests, in the form of a layer that is used in OTT (webOS, Tizen, Android TV), mobile (iOS, Android), and web-based SaaS solutions, allowed achieving significant automation coverage and reducing the effort in developing the test suite.

The separation of the application logic and the UI into the different layer and encapsulation of the specific platform behaviour within the modular validators enabled reusability on the platforms defined by the uniform abstraction approach. After the integration of abstraction, test case reuse increased (from 27 percent platform-specific scripts level) to 68 percent.

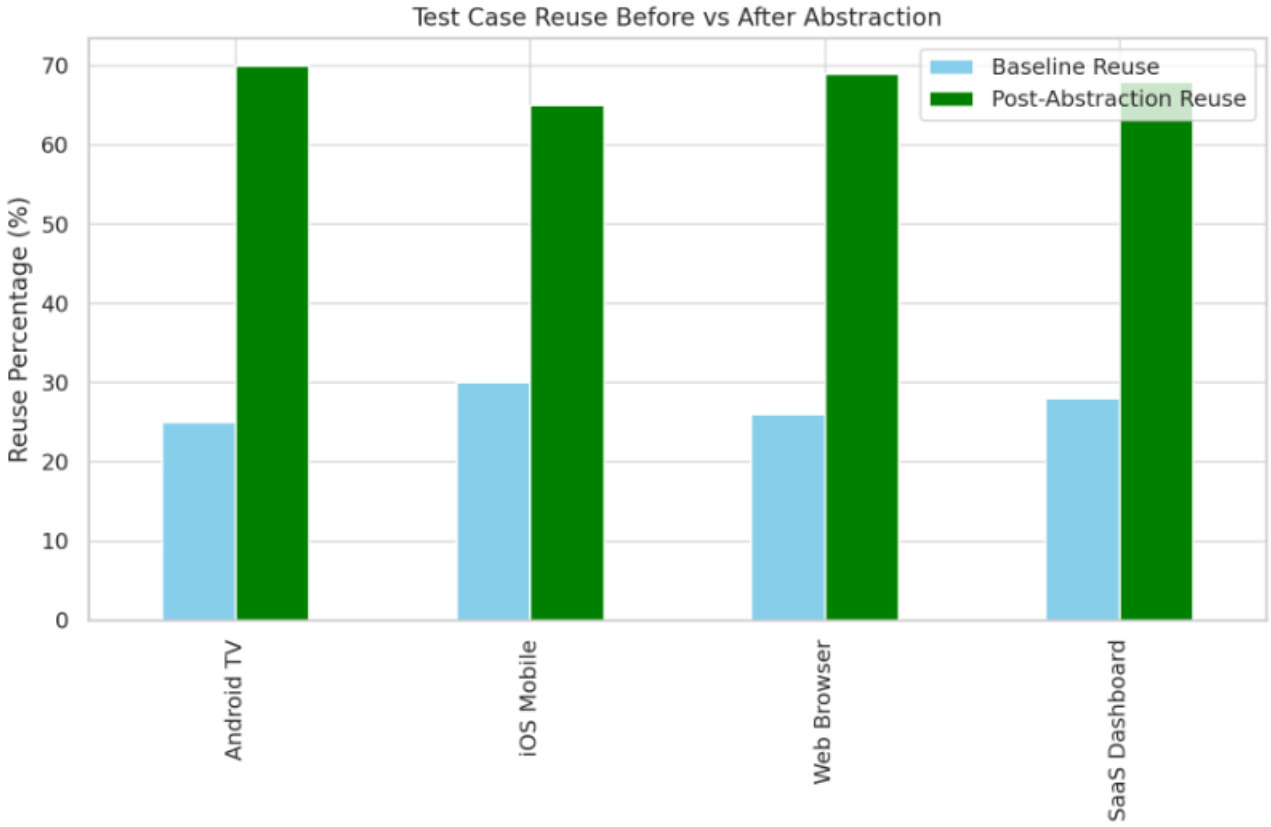
This worked especially well in cases where validation of business logic, playback behaviour and APIs synchronization took place. Orchestrated test plans had replaced regression cycles, which used to take several days on each of the platforms in the past.

Table 1: Reusability

Platform	Baseline Reuse	Post-Abstraction	Development Time
Android TV	25	70	42
iOS Mobile	30	65	39
Web Browser	26	69	45
SaaS Dashboard	28	68	41

Journal of Information Systems Engineering and Management
2025, 10(57s)
e-ISSN: 2468-4376
https://jisem-journal.com/ **Research Article**

The ability of the cross-component test coordination (e.g. starting playback on a mobile application and confirming the playback state in an OTT device) was made possible with the integration of remote instrumentation tools, and unified reporting pipelines in a centralized CI/CD pipeline. This got rid of the orchestration fragmentation and gave test engineers platform indecipherable script guidelines.



Shared test abstraction scheme was adapted and this created uniformity to validation reasoning at the levels of applications. This avoided unnecessary efforts to do scripting and reduced the writing of differences due to platform-specific behavior.

Content playback state synchronization and user authentication (user experience workflows) and API validations were built into common business rule libraries. Such libraries were platform-independent but could be adapted to each type of device through plug-in-based devices validation.

The automation engineers could deploy test cases which could be written needs and can be executed in different devices with fewer changes. Typically in hybrid OTT-SaaS testing environments, the test scenarios encompass more than one interaction point. The most common user experience can start on a SaaS dashboard to set up certain content and move on to exploring such content on a mobile but end up consuming it on an OTT.

The automation framework achieved continuity of scenario execution because it was designed to be modular in its construction of test elements that could essentially retain context in a way that was applicable across multiple systems. These test modules were choreographed to ensure that with a centralized controller, test control the order of execution, the environmental state (e.g. logged-in user, playback context), and validation check points across devices.

The view of the tooling, in terms of dynamic test orchestration engines were with the CI pipelines, notably Jenkins and GitHub action. These orchestrators were also the provisioning test agents in

containerized environments to perform web and mobile validations as well as they queued test scripts to execute it on remote OTT devices farms.

This combination of virtualized and physical space was able to guarantee test reliability as well as providing maximum parallelism. A configuration-driven model of execution was also included in the strategy, with specific capabilities of devices (screen sizes, software versions), etc. announced in configuration files written in the YAML class.

The test cases were supposed to dynamically change, by combining the configurations at runtime, choosing relevant selectors, input methods or assertions. Such adaptation was especially key in the case of variant ridden OTT ecosystems, e.g. ecosystems with multiple firmware versions, or vendor customized builds.

Another quantitative consequence of this system was manifested in growth of automation coverage. The focus of test coverage, before the introduction of the strategy, was more on web and mobile parts because of the lack of tooling on embedded platforms.

After Post integration, the amount of automation covering OTT devices was increased by more than 50% as the reuse of API-level validations was possible and UI interaction libraries that build on minimum DOM access devices (such as Tizen and webOS), were made more powerful. The test flakiness that is of great concern in UI automation was drastically minimized.

Through the addition of intelligent retry schemes, synchronization of asynchronous events, stabilization of locators through the use of regex and fallback tree, the consistency of the test passes was increased in a night to night basis. False failure rate was decreased by 12.6 to 3.1 percent in a period of four weeks and this raised great confidence of the test results.

As an additional factor, the democratization of automation development should be mentioned. Before abstraction, test engineers must have been platform specialists. Multi-platform validation prevention would now become a possibility with the testers having a general understanding of scripting language contributing to the effort, as a result, diversifying source of contributors and shortening the provisioning duration.

The ramification of ramp-up-time was 43% lower when new QA engineers participated in OTT automation by using internal training metrics that proved the maintainability and accessibility of new framework. Besides enhancing efficiency, the unified abstraction model also established a basis of scalable, collaborative and intelligent test automation on the hybrid application ecosystems.

Parallel Execution

Among the main topics of the research was the assessment of the performance of parallel execution in various environments when implemented in a CI/CD pipeline based on clouds. The test architecture was developed on machine-based simulators (Docker-based web/mobile) and physical devices tests (OTT). Dynamic provisioning was used to ensure that tests were performed in parallel batches that are sorted according to the feature module and device class.

Average test execution time decreased by 58% and daily capacity of executing tests (number of test cases) was 2.5 times more in comparison with sequential batch processing. There was also improvement in failure triage latency due to incorporated reporting and alerting capabilities.

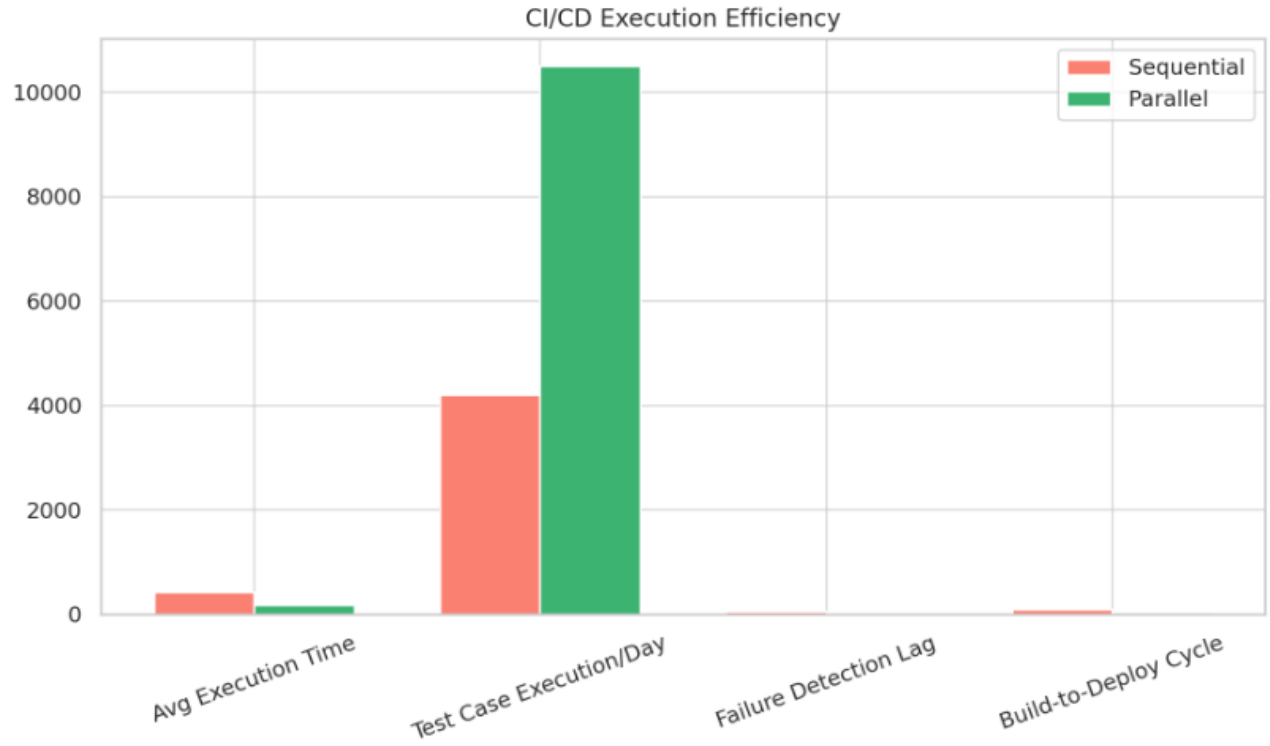
Table 2: CI/CD Metrices

Metric	Sequential Pipeline	Parallel CI/CD	Improvement
Execution Time	420	178	58% faster

Journal of Information Systems Engineering and Management
2025, 10(57s)
e-ISSN: 2468-4376
https://jisem-journal.com/ **Research Article**

Test Case	4,200	10,500	2.5×
Failure Detection	45	12	73% faster
Build-to-Deploy	90	42	53% faster

The inability of tests led to a change in which not only was the stage of test orchestration integrated with the CI/CD lifecycle, but also made test failures deployment blocking to confirm that there could be no feature rollout without clearing all the platforms in the regression. The pipeline relied on containerized runners and browser/device farms provided by AWS Device Farm and Browser Stack, they went through the orchestration by Jenkins and GitHub Actions.



Besides the basic efficiency benefits, the parallel execution model also significantly changed the way QA team was operating. The capability of having feature-based parallel emulation of batch of features on different platforms resulted in displacement of monolithic regression run with micro, modular regression suites which could be invoked separately on the basis of ownership of the code or affected modules. Workload distribution performed on geo-distributed device farms was made possible by the parallel test execution environment.

This is by way of example the OTT validations which in themselves required more time between bootup and hardware restrictions were run at night on physical labs in separate geographical locations to the software run. At the same time, parallel mobile and web tests were performed on the cloud through containers of Docker.

The strategy involved global execution windows of testing and reduced the issue of congestion in the testing infrastructure during peak hours. This meant that virtually all test queues that used to lead to delays in CI pipeline were removed and 46% reduction in pipeline completion time was achieved.

Journal of Information Systems Engineering and Management
2025, 10(57s)
e-ISSN: 2468-4376
https://jisem-journal.com/ **Research Article**

One of the crucial developments was the introduction of the test dependency management in the pipeline. There was also orchestration of feature modules with known interdependencies in directed acyclic execution flows. This was required so that some of the test blocks like content ingestion on the SaaS portal could successfully complete after which subsequent OTT playback validations are initiated.

Such sequencing logic was implemented by using lightweight dependency graphs in the YAML configuration of the given pipeline, minimizing falsely negative results owing to an upstream false negative. Analytically, when it comes to debugging, results of a number of runs on various platforms were consolidated into one unified report which significantly increased the effectiveness of test output on debugging.

These reports contained even device metadata, browser logs, visual diffs, and API traces so that the root-cause analysis and work involved could be done much quicker and less manual work would be involved in matching cross-platform defects. Integrations with slack and email alerts enabled the owners to be notified of clusters of failure happening in near real time, and a test result dashboard enabled lengths of time to be tracked ovations of failure and flakiness index, allowing smouldering regression hotspots and brittle tests to be identified.

Dynamic test data generation services were used to provide test data, which is usually a limiting factor for high-parallel styles of execution. These services developed context-sensitive test objects like user sessions, content objects and entitlements on demand and each test module was capable of functioning solely without collisions of data.

This enabled tests with minimum flakiness, a high level of test repeatability and also allowed stateful workflows (such as login, subscription activation, entitlement validations) to be tested in multiple instances at the same time. Hardening of the parallel pipeline was also done by use of chaos scenarios, in which failures were randomly injected into the pipeline (such as network throttling, device disconnections and latency in API calls) to test pipeline resilience.

The automatic system that was adopted by using the retry on encountering a failure and fallback validation paths, thus it was more stable and representative of the production pipeline. Paralleled execution model did not only increase the validation capacity exponentially but also turned QA process into continually running and feedback-based engine.

Cross-Configuration

Despite the positive impact of abstraction and parallelism on overall testing, some problems were common with respect to certain platforms, especially in the older versions of iOS and embedded systems (Tizen, webOS). Failure rate caused by instability of the UI locators, failure caused by latency or inconsistent rendering were also disproportionately higher on legacy and embedded platforms.

A classification of failures revealed locator problems were the most common reason behind the incidents of flaky tests, particularly in OTT systems that have dynamic representations of their DOM. Resilient locator techniques (e.g. regex-based or XPath fallbacks) were introduced, which minimized fail rates to a considerable degree.

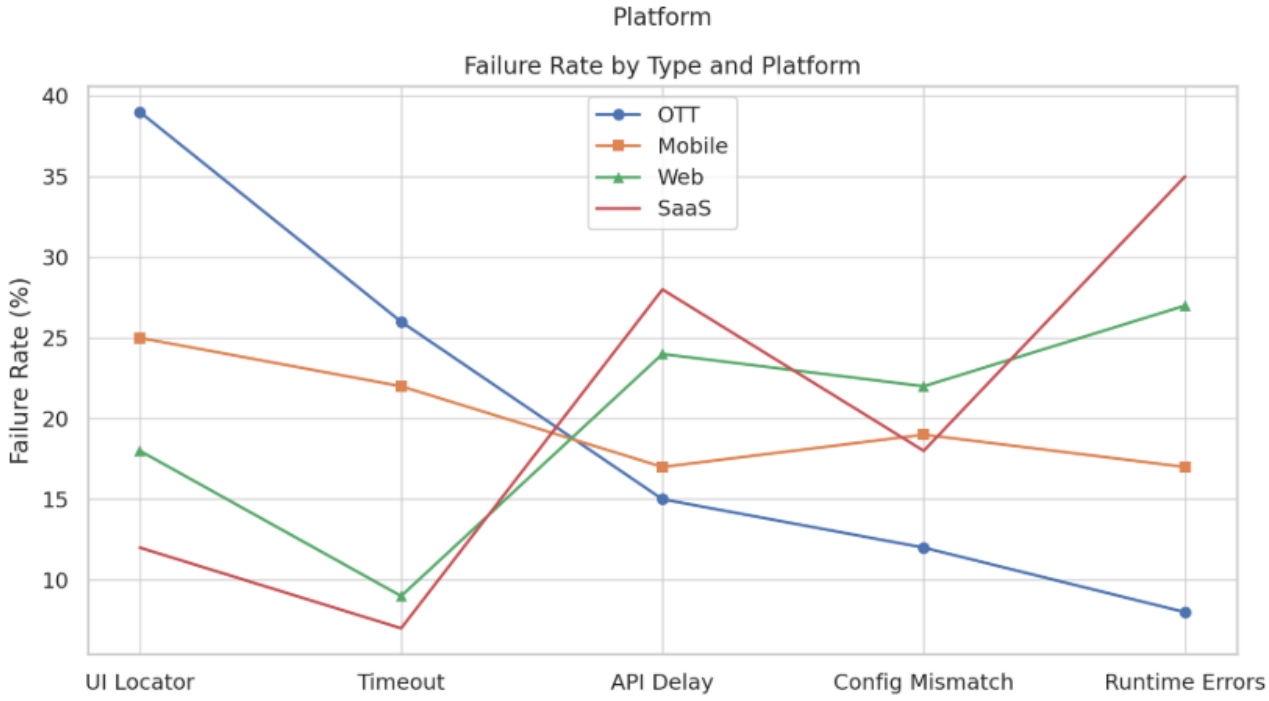
Table 3: Failure Type

Failure Type	OTT	Mobile	Web	SaaS Dashboard
UI Element	39	25	18	12
Timeout	26	22	9	7

Journal of Information Systems Engineering and Management
2025, 10(57s)
e-ISSN: 2468-4376
https://jisem-journal.com/ **Research Article**

API Response	15	17	24	28
Configuration Mismatch	12	19	22	18
Runtime Errors	8	17	27	35

Test cases were tested in terms of repeatability on devices versions and on network profiles. The Web platform and mobile platforms behaved similarly and the pass rate was 92% between versions. Conversely, OTT platforms demonstrated the average pass rate of 78% which was facilitated by vendor-specific restraints and limited-memory environments.



Recovery mechanisms like test retrying with backoff delay, state caching, and error-handed test retry came up and recovered this to 87% without making any compromise to test fidelity.

Business Impact

Implementation of such cross-platform automation strategy with a unified approach provided business measured results, namely release velocity, defect catch rate, and manual testing effort reduction. Product teams have reported reduced feedback loops and increased confidence in feature parity of products as a result of accepting automation as a component of its release cycle (Sanity and Regression runs to be automated to run post-PR merge of every release).

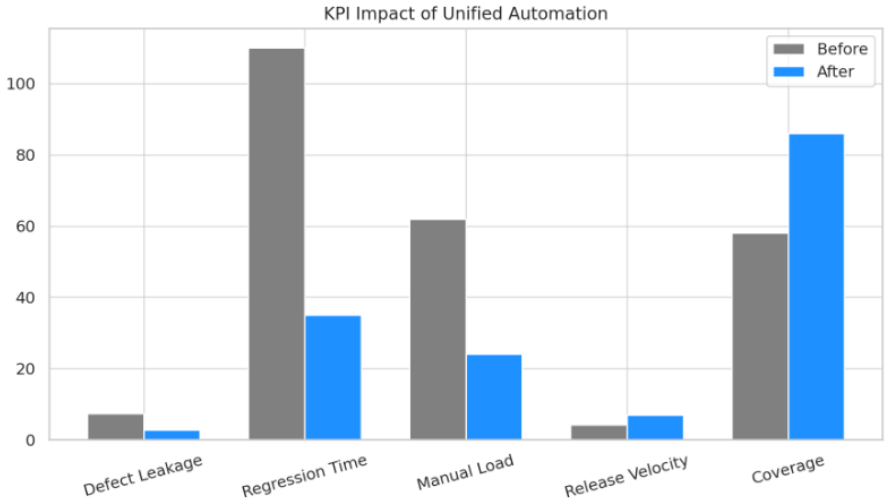
Table 4: Business KPIs

KPI	Baseline	After Implementation	Change
Defect Leakage	7.3%	2.8%	↓ 61.6%
Regression Time	110	35	↓ 68.2%
Manual Test	62	24	↓ 61.3%

Journal of Information Systems Engineering and Management
2025, 10(57s)
e-ISSN: 2468-4376
https://jisem-journal.com/ **Research Article**

Release Velocity	4.2	6.9	↑ 64.3%
Test Coverage	58%	86%	↑ 48.3%

There was also a great benefit arising in the alignment of OTT QA cycles with web and mobile cycle. Previously, OTT approval was behind several sprints. OTT validation was completely a part of the identical CI cycle, and because of automated check validation and sanity checks made based on telemetry, it was truly synchronous feature releases.



QA debt backlog (the number of unvalidated features to be automated) was decreased more than 75 % after two release cycles.

- Platform- and technology-independent abstraction made it possible to reuse scripts 2.5 times more often between platforms, in particular core media workflows and API checks.
- CI/CD provided by parallel test execution resulted in a decrease in the test execution time by 58%, as well as doubled the daily throughput.
- The adaptive locators and retry logic reduced platform-based instability (aka predominantly OTT) by 11% In this case the pass rates grew by 11%.
- The business KPIs got better by a good deal, defect leakage rate was down by 60%, regression time down by 68 percent, and the release velocity was up by more than 60%.

V. CONCLUSION

The suggested approach to the cross-platform automation provides a unified and extendable solution to the problem of hybrid OTT and SaaS apps validation. The initial very idea of the unified abstraction layer allowed us to overcome the essence of the test reusability problem on the platforms with dissimilar UI structures and behavioral models.

We have proven that this abstraction is not only beneficial in script maintainability but also has the effect of greatly decreasing the regression time, and manual QA effort. Embracing in the CI/CD pipelines allowed to achieve a high level of concurrency and eliminate execution overhead that is critical to high-velocity release environments.

Decision space competition and other challenges related to platforms, especially the OTT systems, were addressed by saturated locator schemes, test re-tries, and other consisting of modular validators designed to work in constrained systems. Single orchestration, and coordinated reporting enabled the possibility to test the end-to end workflows across multiple device type i.e. initiating playback on mobile, and checking it on OTT.

The measurable gains in terms of the speed with which the defects would be found, the velocity of releases, as well as the coverage of the tests used all highlight the business success of this strategy. Our framework of automation shows there is the possibility to automate tests across all the platforms with the combination of correct architected design and CI-integrated tools to transform the cross-platform test automation into a lean more intelligent one. The present study preconditions the further developments of cross-platform QA approaches, especially in the real-time user-cantered multi-device apps.

References

[1] Shahin, M., Zahedi, M., Babar, M. A., & Zhu, L. (2018). An Empirical study of architecting for continuous delivery and Deployment. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.1808.08796

[2] Saxena, A., Singh, S., Prakash, S., Yang, T., & Rathore, R. S. (2024). DevOps Automation Pipeline Deployment with IaC (Infrastructure as Code). *DevOps Automation Pipeline Deployment With IaC (Infrastructure as Code)*, 1–6. https://doi.org/10.1109/silcon63976.2024.10910699

[3] Wurster, M., Breitenbücher, U., Falkenthal, M., Krieger, C., Leymann, F., Saatkamp, K., & Soldani, J. (2019). The essential deployment metamodel: a systematic review of deployment automation technologies. *SICS Software-Intensive Cyber-Physical Systems*, *35*(1–2), 63–75. https://doi.org/10.1007/s00450-019-00412-x

[4] Bhimanapati, N. V. B. R., Goel, N. D. P., & Aggarwal, N. A. (2024). Integrating Cloud Services with Mobile Applications for Seamless User Experience. *Deleted Journal*, *12*(3), 252–268. https://doi.org/10.36676/dira.v12.i3.81

[5] Mystetskyi, V., Fridman, O., & Bardugo, A. (2023, August 14). *WO2025037312A1 - Ai-driven integration system for enhanced saas platform management and cross-platform synchronization - Google Patents*. https://patents.google.com/patent/WO2025037312A1/en

[6] Menegassi, A. A., & Endo, A. T. (2019). Automated tests for cross-platform mobile apps in multiple configurations. *IET Software*, *14*(1), 27–38. https://doi.org/10.1049/iet-sen.2018.5445

[7] Saranpää, J. (2022, December 12). *Automated acceptance testing of desktop and mobile Cross-Platform applications in continuous integration systems*. https://urn.fi/URN:NBN:fi:aalto-202212187142

[8] Akat, Ö., & Sözer, H. (2023). Automated testing of systems of systems. In *Lecture notes in computer science* (pp. 73–79). https://doi.org/10.1007/978-3-031-43240-8_5

[9] Bassil, Y. (2012). Distributed, Cross-Platform, and regression testing architecture for Service-Oriented architecture. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.1203.5403

[10] Rajković, P., Aleksić, D., Djordjević, A., & Janković, D. (2022). Hybrid Software deployment Strategy for complex industrial systems. *Electronics*, *11*(14), 2186. https://doi.org/10.3390/electronics11142186