ISSN 2278-3091
Volume 14, No.6, November – December 2025
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse051462025.pdf
https://doi.org/10.30534/ijatcse/2025/051462025

End-to-End (E2E) Testing Methodology as a Tool for Managing the Digital Product Lifecycle and Minimizing Economic Losses at the Operational Stage

Yulia Drogunova
Bachelor's degree, Dostoevsky Omsk state university, Russia, Omsk

Received Date: October 24, 2025 Accepted Date: November 27, 2025 Published Date: December 06, 2025

ABSTRACT
The article explores the end-to-end (E2E) testing methodology as a tool for managing digital products and reducing economic losses during their operational phase. The role of the E2E approach is analyzed across various stages of software development and maintenance – from design to support. Particular emphasis is placed on the reproduction of user scenarios as a means of detecting defects that are not covered by other forms of testing. The effectiveness of E2E automation is substantiated within the context of CI/CD and DevOps practices. As an example of industrial implementation, the architectural strategy adopted by Uber is examined. The article concludes that E2E testing represents a strategically important mechanism for mitigating business risks, stabilizing release cycles, and optimizing the total cost of ownership of digital products.

Key words : End-to-end testing, E2E testing, product lifecycle, digital product, automation, CI/CD, business risks.

1. INTRODUCTION

In the context of digital transformation of the economy, the resilience and predictability of software systems have become critical factors for business continuity. Even minor defects during the operational phase of a digital product can lead to substantial economic losses, disruptions in business processes, and a decline in user trust. In this regard, end-to-end (E2E) testing is gaining particular relevance as it enables validation of software functionality under conditions that closely resemble real-world usage. Unlike localized forms of testing, E2E focuses on verifying the entire chain of interactions between components, services, and user scenarios, which facilitates early defect detection and reduces the likelihood of failures in production environments.

The purpose of this article is to examine the E2E testing methodology as a means of managing the digital product lifecycle and mitigating business risks associated with its operation.

2. E2E TESTING IN THE CONTEXT OF THE DIGITAL PRODUCT LIFECYCLE

The lifecycle of a digital product encompasses a set of interrelated stages that cover the full trajectory of a software system – from requirements gathering and architectural design to development, testing, deployment, maintenance, and subsequent modernization or decommissioning (figure 1).

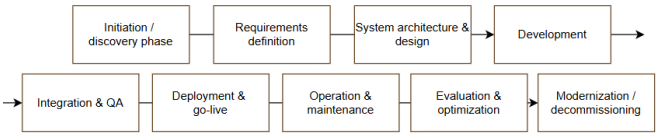

Figure 1: Digital product lifecycle

In general, E2E testing represents a verification methodology aimed at comprehensively validating the correct functioning of a digital product as it passes through all layers of architecture and interaction – from the user interface to databases, external APIs, and business logic [1]. It makes it possible to simulate real user scenarios within a test environment and assist in the discovery of errors that are not the result of individual components themselves, but are generated by their interaction on E2E business processes. Thus, E2E testing provides better validation reliability and plays a crucial role in attaining the quality of digital solutions. Table 1 presents an overview of the role of E2E testing at various stages of the digital product lifecycle.

Table 1: Role of E2E testing at different stages of the digital product lifecycle

Lifecycle stage	Role of E2E testing
System architecture & design	Definition of end-user scenarios; embedding testability into the architecture and design artifacts.
Development	Implementation of basic E2E tests in parallel with feature development; validation of logical flows and dependencies.
Integration & QA	Primary stage for E2E application: verifying the integrity of business processes and cross-component interactions.
Deployment & go-live	Final validation before release; execution of smoke and sanity tests to ensure system readiness.

Operation & maintenance	Ensuring stability during updates and patches; prevention of regressions in the production environment.

The presented table illustrates that E2E testing is applied not at a single point, but across multiple critical stages of the digital product lifecycle. Its implementation ensures continuous control over the integrity of user scenarios and minimizes the risk of regressions during transitions between development and operational phases.

Unlike unit testing, which focuses on individual functions, and integration testing, which addresses interactions between subsystems, E2E testing reproduces real user scenarios while taking into account all relevant factors of the execution environment (table 2).

Table 2: Comparative analysis of testing types within the digital product lifecycle [2, 3]

Parameter	Unit testing	Integration testing	E2E testing
Test objective	Validate individual functions or methods in isolation.	Verify interactions between integrated components.	Confirm full system behavior from the user's perspective.
Scope	Isolated code blocks	Interfaces and data exchange between modules.	Entire system: UI, APIs, services, and databases.
Defects detected	Logical or syntax errors within single units.	Interface mismatches, data format issues.	Workflow failures, regressions, cross-component inconsistencies.
Execution environment	Fully isolated with mocked dependencies.	Partial system with real services.	Near-production environment with real data and services.
Speed and cost	Fast and low-cost due to minimal setup.	Moderate setup and execution time.	Resource-intensive due to complexity and full-stack execution.

In practice, the E2E testing methodology is closely integrated into CI/CD pipelines, enabling automated validation of product integrity after each release. Within the product lifecycle, this means that E2E tests operate as a mechanism of continuous quality control, providing feedback at every stage of change. Moreover, as is the case in DevOps culture and high-frequency release cycles, E2E testing has a critical role in reducing time-to-production alongside enhancing system stability. Thus, E2E testing not only has a technical role but also a managerial role to foster alignment and continuity across different stages in the digital product lifecycle.

2. E2E TESTING METHODOLOGY

Within the E2E testing methodology, test case modeling is understood as a formalized process of constructing digital scenarios that reflect the complete behavior of the user when interacting with the system. This approach requires not merely the creation of isolated checks, but the sequential description of business process logic under conditions that closely resemble real-world operation. Methodologically, it is essential that each modeling stage ensures traceability to business requirements, resilience to changes in the execution environment, and suitability for automated execution within CI/CD pipelines (table 3).

Table 3: Stages of test case modeling in E2E testing methodology [4, 5]

Modeling stage	Description
1. Identification of user scenarios	Selecting key business processes for testing based on system requirements, risk analysis, and user flows.
2. Definition of preconditions	Specifying the initial system state: environment setup, test data, user roles, and authentication context.
3. Specification of expected outcomes	Clearly defining success criteria: interface states, API responses, data changes, and business validations.
4. Step-by-step scenario construction	Describing each user interaction, including navigation, inputs, system responses, and assertions.
5. Handling external dependencies	Managing instability via mocks, stubs, and fixtures to ensure test determinism and environmental control.
6. Verification and traceability	Ensuring alignment of the test case with business requirements; linking to specifications or user stories.
7. Preparation for automation	Structuring the scenario for implementation in tools such as Cypress, Selenium, or Playwright.

In practice, the automation of E2E tests is implemented using specialized frameworks that support user scenario execution, manage browser environments, and integrate with CI/CD pipelines. The most widely used tools include Selenium, Cypress, and Playwright – each with its own architectural characteristics, application domains, and limitations. The choice among them depends on factors such as browser compatibility requirements, performance expectations, level of access to the DOM, and the configurability of the execution environment (table 4).

Table 4: Comparative overview of E2E testing tools [6, 7]

Parameter	Selenium	Cypress	Playwright
Execution context	External driver controlling real browsers.	Executes inside the browser with direct access to the DOM.	External control with built-in browser support and network management.
Browser support	Chrome, Firefox, Safari, Edge, IE.	Chromium-based only (Chrome, Edge).	Chromium, Firefox, WebKit (including mobile emulation).
Programming languages	Java, Python, C#, JavaScript, Ruby.	JavaScript, TypeScript.	JavaScript, TypeScript, Python, C#.

Parallel execution	Supported via external configuration (e.g. Selenium Grid).	Limited, experimental via third-party tools.	Native support for parallel test execution.
Speed and performance	Moderate, depends on driver/browser communication overhead.	High, due to in-browser execution.	High, with granular control over timeouts and resources.
Parameter	**Selenium**	**Cypress**	**Playwright**
Access to network/API	Indirect, via additional libraries or test code.	Limited; requires plugins for stubbing API.	Native API request interception and mocking.
Test stability	Medium (requires manual synchronization in complex flows); sensitive to asynchronous behavior and timing issues.	High; built-in automatic waits and retries.	High; includes context isolation and precise control over execution timing.
Reporting tools	Integrates with Allure, TestNG, JUnit, etc.	Built-in dashboard, Allure via plugins.	Built-in reporters, supports integration with Allure and HTML reporters.
CI/CD integration	Jenkins, GitLab CI, GitHub Actions, Azure DevOps.	Seamless integration via npm and CLI.	Built-in support for most CI/CD pipelines.

Thus, the effectiveness of the E2E testing methodology largely depends on the quality of user scenario modeling and the rational selection of automation tools. A well-chosen technology stack not only ensures comprehensive functional coverage and a high degree of test reproducibility but also enables the integration of E2E quality control into the CD process of the digital product. Collectively, this makes E2E testing an essential component of engineering quality assurance practices throughout all stages of the software system lifecycle.

3. E2E TESTING AS A MECHANISM FOR RISK MANAGEMENT AND ECONOMIC RELIABILITY OF DIGITAL PRODUCTS

In the phase of industrial operation, defects that were not identified during earlier stages of verification become direct sources of economic loss. Unlike bugs discovered during development, production faults affect not only the user experience but also the continuity of business processes, integration stability, SLA adherence, and the commercial stability of the system as a whole. It is particularly critical for high-load services, financial systems, and enterprise information systems, where even brief disruptions result in reputational losses and legal consequences.

Typical errors that arise in production due to insufficient test coverage include:

- violations of business logic,
- authorization failures,
- errors in multi-step forms,
- data loss during state transitions,
- version mismatches between frontend and backend components.

A particularly serious threat is posed by defects that occur only under complex combinations of conditions – such as specific sequences of actions, differences in user roles, unstable network latency, or concurrent data access. These types of failures are typically beyond the detection capabilities of unit or integration tests, as they require full reproduction of user flows that traverse multiple layers of the system.

The E2E testing methodology is designed to detect precisely those high-risk, low-predictability errors. E2E tests simulate real user behavior by examining the system from the outside – through the user interface and all intermediate layers, down to databases and external APIs. This depth of coverage enables verification not only of technical correctness but also of the logical consistency of business operations. The integration of E2E testing into CI/CD pipelines allows for automatic validation of user scenario stability with every code, configuration, or environment change, thereby significantly reducing the risk of introducing regression defects into the production environment.

A notable example of a systematic approach to E2E testing can be found in the architecture of Uber, which has implemented the internal initiative known as BITS (Backend Integration Testing Strategy). This strategy enables automated verification of every code and configuration change across more than 1,000 backend services within the company. Under this framework, thousands of E2E tests are executed to cover critical user scenarios – for instance, the process of placing a group order in Uber Eats. These tests are run in isolated sandbox environments, allowing for parallel validation of individual commits without compromising the stability of the broader system. Notably, up to 30 % of incidents at Uber have been attributed to configuration errors, and the BITS framework provides test coverage even for such changes, effectively preventing them from reaching the production environment (figure 2).

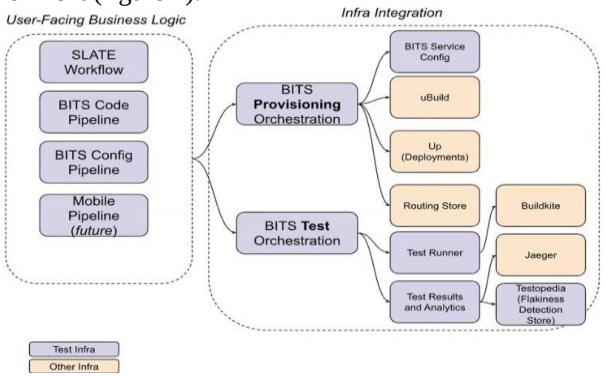

Figure 2: Architectural components of BITS [8]

Each execution of an E2E test within this framework is accompanied by trace collection using the Jaeger™ tool, which generates coverage indices across services and endpoints. This mechanism not only ensures transparency but also enhances test stability: while individual tests may initially pass with a probability of ≥ 90 %, repeated executions can raise this success rate to 99,9 %. In Uber's largest service alone, more than 300 E2E tests are executed, and even with a per-run success probability of 95 %, the system effectively minimizes cumulative failure risk. As such, integration-level E2E testing becomes a powerful risk management tool – addressing direct economic losses caused by regression defects, SLA violations, and release rollbacks. The repeatability of these processes across thousands of daily test runs demonstrates that, with a well-architected infrastructure, E2E testing can serve not only as a reliable quality control mechanism but also as a critical component of the economic resilience of a digital platform.

It should be noted that E2E testing plays a significant role in establishing a stable and economically efficient model for operating a digital product. Its implementation not only enhances the stability of user scenarios but also systematically reduces costs associated with maintenance, defect resolution, and ensuring the reliability of the release cycle (table 5).

Table 5: Impact of E2E testing on reducing economic losses in production [9, 10]

Factor	Economic effect of E2E implementation
Late detection of defects	Cost reduction through early identification of critical issues before production deployment.
Post-release regressions	Prevention of high-severity incidents and rollbacks; stabilization of release cycles.
Incident resolution time (MTTR)	Increased responsiveness and faster mitigation of production failures.
Volume of manual testing	Optimization of QA efforts through automation of repetitive user flows.
Reliability of delivery pipeline	Improved release predictability and reduced likelihood of critical deployment errors.
Total cost of ownership (TCO)	Lower operational and maintenance expenses throughout the product lifecycle.
Return on investment (ROI) in automation	Higher effectiveness of QA infrastructure and long-term benefits from early testing integration.

Thus, E2E testing functions not merely as a quality assurance tool, but as a strategic mechanism for risk management, reduction of operational costs, and ensuring the economic reliability of a digital product. In the context of increasing architectural complexity, stricter requirements for fault tolerance, and shorter release cycles, its application becomes an integral component of mature engineering and management practices.

5. CONCLUSION

The E2E testing methodology constitutes a systematic approach to the verification of digital products, spanning the entire lifecycle – from architectural design to production maintenance. Unlike isolated testing techniques, E2E ensures the validation of complete user scenarios under conditions close to real-world operation. This enables the detection of defects arising from component interactions, reproduction of critical process paths, and confirmation of compliance with business logic. The adoption of E2E testing enhances not only the technical resilience of a product but also the controllability of development and delivery processes, fully integrating with DevOps culture and CI/CD practices.

From an economic perspective, E2E testing significantly reduces the TCO by enabling early defect detection, stabilizing the release cycle, and lowering support and maintenance costs. Automation of E2E tests ensures reproducibility, reduces manual testing overhead, and mitigates the risk of incidents in production. Against the backdrop of increasingly complex digital platforms and high reliability demands, E2E testing emerges not only as a quality control practice but as a vital instrument for business risk management – supporting cost reduction and fostering trust in the digital product throughout its entire lifecycle.

REFERENCES

1. S. Di Meglio, L. L. L. Starace, S Di Martino. **E2E-Loader: A Tool to Generate Performance Tests from End-to-End GUI-Level Tests,** *2025 IEEE Conference on Software Testing, Verification and Validation (ICST). IEEE,* pp. 747-751, 2025.
2. M. A. G. Elgazzar, M. E. T. Dawood. **Mastering the Phases of Design Strategy: A Comprehensive Guide to Navigating from Conceptualization to Launch in the Digital Product Life Cycle,** *Journal of Art, Design and Music*,vol. 4, no. 2, pp. 1, 2025.
3. A. Smirnov. **Methods for detecting and resolving issues in API: profiling and performance optimization,** *Cold science*, no. 14, pp. 7-15, 2025.
4. D. Olianas, M. Leotta, F. Ricca. **BEWT: A Benchmark for End-to-End Web Testing,** *Euromicro Conference on Software Engineering and Advanced Applications. Cham: Springer Nature Switzerland*, pp. 296-314, 2025.
5. R. Garifullin. **Analysis and development of interfaces for emergency systems with multitasking processing: approaches to minimizing user errors and enhancing reliability,** *International scientific journal «Innovative Science»*, no. 1-2-1, pp. 20-24, 2025.
6. M. Leotta, B. García, F. Ricca, J Whitehead. **Challenges of end-to-end testing with selenium webdriver and how to face them: A survey,** *2023 IEEE Conference on Software Testing, Verification and Validation (ICST). IEEE*, pp. 339-350, 2023.
7. S. Talakola. **Automated end to end testing with Playwright for React applications,** *International Journal of Emerging Research in Engineering and Technology*, vol. 5, no. 1, pp. 38-47, 2024.
8. Shifting E2E Testing Left at Uber / Uber // URL: https://www.uber.com/en-AT/blog/shifting-e2e-testing-left/ (date of application: 25.09.2025).

9. B. Zendeli, A. Luma, B. Selimi, B. Rexha. **Identifying Limitations in End-to-End Testing: A Systematic Review of Coverage Analysis Techniques,** *2025 7th International Congress on Human-Computer Interaction, Optimization and Robotic Applications (ICHORA). IEEE*, pp. 1-9, 2025.
10. R. Zhao, S. Zhang, Z. Zhu, Y. Shang, W. Wang. **E2E test execution optimization for web application based on state reuse,** *Journal of Software: Evolution and Process*, vol. 37, no. 1, pp. e2714, 2025.