Journal of Advances in Developmental Research (IJAIDR)

E-ISSN: 0976-4844 ● Website: www.ijaidr.com ● Email: editor@ijaidr.com

Dynamic Test Case Generation for External Transfer APIs Using Java TestNG and Apache POI in Legacy-to-Cloud Migrations

Udayan Verma

Denver, USA
udayanverma7@gmail.com

Abstract:
This paper proposes an active test case generation model meant to check External Transfer APIs in the projects of legacy to cloud migration. Financial systems that handle high-value transactions are distributed across various fields and frequently use heterogeneous systems, which complicate the validation process. These environments cannot be addressed with manual testing and static test suites due to their inability to scale to changing business rules or support large amounts of test data. The solution proposed uses Java TestNG to coordinate parallel test runs, conditional logic and assertions, and Apache POI to dynamically read structured datasets in Excel files. These are datasets that consist of account details, type of transactions and anticipated results and are converted to test scenarios. The framework allows the creation of tests at any scale and without hard coded values, as well as field-level validation in both the cloud and legacy environment. Reporting is done using HTML and XML dashboards which gives traceable results and has integration with defect tracking systems. The methodology minimizes human work, shortens regression cycles and creates an auditable and repeatable testing platform on critical financial API migrations.

Keywords: Dynamic Test Case Generation, Financial API Validation, Data-Driven Testing, Legacy-to-Cloud Migration, TestNG and Apache POI.

I. Introduction

The process of migrating External Transfer APIs that handle high-value transactions between the legacy systems and cloud infrastructures creates significant challenges to the financial system validation. Mistakes in data mapping, schema compliance, or business rule implementation may result in financial inaccuracies, regulatory problems, and customer loss. Traditional off-the-shelf test suites are not up to these demands because they are not scalable to large volumes of test data or to complex business transactions. The Java TestNG in combination with Apache POI were designed as a dynamic test case generation model of Charter Communications. It reads structured Excel data to create tests dynamically which minimizes hardcoding, error, and allows cross environment verification. The framework provides alignment of cloud APIs with legacy records field-by-field providing a repeatable, auditable, and efficient method of financial migration testing.

II. Background

The process of legacy-to-cloud migration is complicated by the fact that financial institutions and large enterprises use many heterogeneous systems. It is also vital to maintain transactional consistency between the old and new infrastructures because any simple mistake can have serious economic and regulatory implication. Manual testing takes too long, is prone to error, testing coverage on high-value transactions are not sufficient [1]. Automated manual test suites can be static, but lack the ability

transactional systems adapt to continuous business rules or complexity of cloud systems. Data driven test case generation provides a method. Using structured records to generate test case algorithms simulates real-world business functions without hard-coded data. This improves scalability, accuracy, and regression speed. With automated orchestration and dynamic inputs, financial APIs are verified seamlessly and reliably across systems.

III.Framework Architecture and Design

The dynamic test case generation framework is designed to efficiently handle complexity and still retain scalability. It starts by ingesting structured Excel files via Apache POI which represent accounts, transactions, and expected outcomes [2]. This data is parsed into test parameters to create multiple scenarios without manual scripting before processing test parameters once. This provides for efficiency and scalability due to being able to lose volume in transactional loads.

TestNG then dynamically creates tests when it is processed. The framework allows conditional execution and parallel processing where many scenarios can be executed at the same time, minimizing the regression cycle time and making sure that different transactions are checked within narrow windows. Field-level testing is conducted in the legacy and in the cloud environment where each transaction is verified to be as expected.

Reporting is built in with HTML and XML dashboards, which give detailed information on test results. Deviations automatically appear in defect tracking systems, which help trace and audit testing. The modular structure improves maintainability and flexibility of the framework with the ability to update test cases through modification of Excel data without changing the fundamental code [3]. This flexibility means that the system will be able to effectively adapt to changing business policies and migration needs, allowing scalable and repeatable financial API validation.

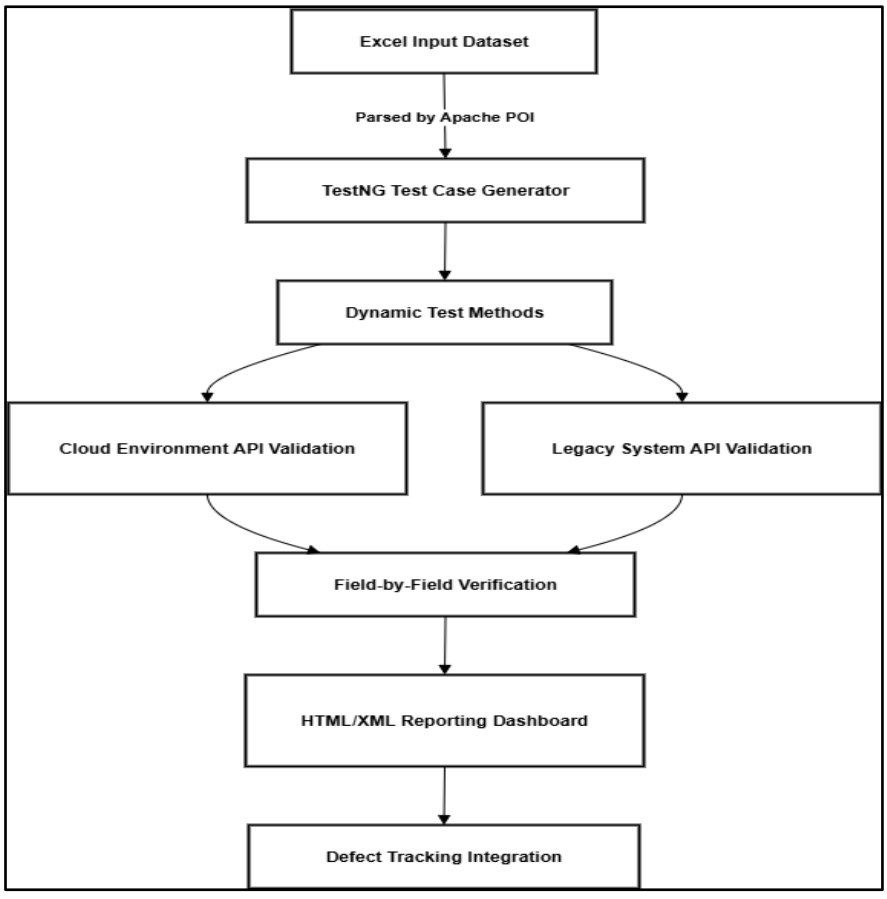

Fig. 1: Framework Architecture Diagram

IV. Implementation Details

The foundation of the framework begins with structured Excel datasets as a operation of legitimate transactional scenarios. Each row contains the account identifiers, transaction type, the amounts, and expected outcomes [4]. Apache POI is used to parse each of these datasets and convert from raw data to objects that can be executed for testing, so that the framework can test thousands of test cases without human intervention.

TestNG will dynamically create test method based on the data parsed. Each test, performs API calls, checks the response, and compares expected outcomes. Each transaction type has condition logic applied to make sure that all business rules are followed when executing. Parallel execution of tests optimizes resource utilization and execution time, allowing a wide range of test cases to run simultaneously.

Reporting features are built in to provide useful feedback. Results and evaluations are presented in HTML and XML dashboards that show pass and fail rates, execution times and detail per field that failed. These dashboards are exportable and compatible with defect management tools in order to have full audit functionality [5]. The modular framework allows for test cases to be updated or added to simply by using the Excel datasets only without making any change to the core code. This keeps the framework scalable, maintainable, and adaptable to new business rules, requirements, or more complex transaction levels in order to reliably validate the financial APIs.

Table 1: Sample Excel Input Dataset

Account ID	Transaction Type	Amount (USD)	Expected Result
100001	Transfer	5000	Success
100002	Transfer	15000	Insufficient Funds
100003	Transfer	2000	Success
100004	Transfer	0	Invalid Amount
100005	Transfer	7500	Success
100006	Transfer	30000	Exceeds Daily Limit
100007	Transfer	1200	Success
100008	Transfer	-500	Invalid Amount
100009	Transfer	10000	Success
100010	Transfer	500	Success

V. Scalability and Optimization

The framework is architected to scale efficiently for the high volume of financial transactions typically encountered during migration projects. Because parallel execution is used, test scenarios are able to execute in concurrent fashion, minimizing regression cycle time and accommodating large volume API validation in constrained time windows. Test case updates occur simply through changing Excel datasets, or creating additional Excel datasets, allowing updates of new transaction types, or changes in business rules, without impacting core code [6]. TestNG's parallel execution optimizes resource use by running multiple threads simultaneously, enabling fast regression cycles without losing accuracy. Its cloud integration adds scalability, ensuring efficient, adaptable testing with quick feedback for confident API migrations.

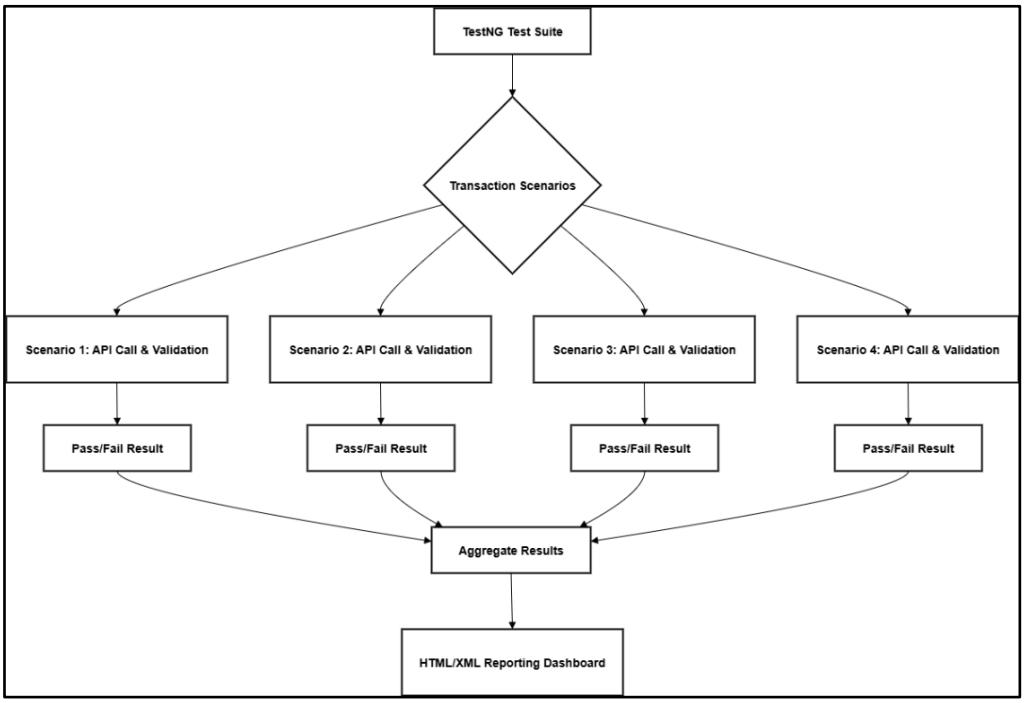

Fig.2: Parallel Execution Workflow

VI. Reporting and Monitoring

Reporting and monitoring make test results actionable and traceable. The framework offers detailed HTML and XML dashboards on pass rates, execution time, and field-level discrepancies. Failures are consistently captured via integration with defect tracking, thereby enabling auditability and traceability [7]. Historical data analysis supports the identification of recurring problems and improvement opportunities that provide better overall testing efforts. This methodology benefits users with visibility on test results, and a way for them to act quickly to problems to help retain quality throughout financial, API migrations.

VII. Conclusion and Future Work

The dynamic test case generation framework is for External Transfer APIs is a reliable and scalable method of validating financial transactions in legacy to cloud migrations. This framework uses Java TestNG and Apache POI to provision test cases, validates business rules and acceleration of regression cycle time via parallel execution. Structured Excel datasets provide flexibility in updates via controlling Excel datasets. Integrated reporting and audit tracking ensures identifiable traceability, auditability, and actionable testing, so users can understand testing results. Future improvements may involve AI-driven test data generation, predictive regression prioritization, and cloud-native orchestration to boost scalability and efficiency, ensuring a robust and reliable testing process for critical financial AP migration.

REFERENCES:

1. Arcuri, A., "RESTful API Automated Test Case Generation," *2017 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pp. 9-20, 2017.
2. Scalabrino, S., Bavota, G., Linares-Vásquez, M., Lanza, M. and Oliveto, R., 2019, May. Data-driven solutions to detect api compatibility issues in android: an empirical study. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)* (pp. 288-298). IEEE.
3. Qi, L., He, Q., Chen, F., Zhang, X., Dou, W. and Ni, Q., 2020. Data-driven web APIs

recommendation for building web applications. *IEEE transactions on big data*, *8*(3), pp.685-698.

4. Althani, B. and Khaddaj, S., 2017, October. Systematic review of legacy system migration. In *2017 16th International Symposium on Distributed Computing and Applications to Business, Engineering and Science (DCABES)* (pp. 154-157). IEEE.
5. Ge, X., Yu, S., Fang, C., Zhu, Q. and Zhao, Z., 2022. Leveraging android automated testing to assist crowdsourced testing. *IEEE Transactions on Software Engineering*, *49*(4), pp.2318-2336.
6. Kiranagi, V.H. and Shyam, G.K., 2017, August. Feature Driven Hybrid Test Automation Framework (FDHTAF) for web based or cloud based application testing. In *2017 International Conference On Smart Technologies For Smart Nation (SmartTechCon)* (pp. 1555-1559). IEEE.
7. Ramlall, S., Gutierrez, L., Cayetano, M. and McGehee, S., 2022, August. A Modular Approach to the Automated Testing of Common Data Link Terminals. In *2022 IEEE AUTOTESTCON* (pp. 1-6). IEEE.