Journal of Advances in Developmental Research (IJAIDR)

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

REGRESSION ISOLATION FOR UI/API BUGS USING XPATH-DRIVEN OBJECT SNAPSHOTTING AND HISTORICAL PAYLOAD COMPARISON

Udayan Verma

Denver, USA
udayanverma7@gmail.com

Abstract:
The report defines a current, evidence based, model of quality assurance (QA) of multi-layered and microservice based systems. The human-based problem which is being targeted is the fault identification. A set of rightful attribution of failures, tracking of changes in the baseline, automatic generation of reports of revision changes, and continuous integration/ continuous deployment (CI/CD) is used to offer a powerful solution. It is based on object capture and historical payload baseline on XPath, which can be used to enable the fastest recognition of regressions, and it wastes less time in performing the exploration and testability between User Interface (UI) and Application Programming Interface (API) layers. The proposed plan to arrange the software architecture and pipeline frame helps in providing the evidence-based systematic debugging plan that is required in the process of maintaining the quality of software in agile systems.

Keywords: Failure Attribution, Baseline Monitoring, Change Management, Continuous Integration, Test Automation.

INTRODUCTION

Modern agile development and fast release cycles demand updated QA approaches. Continuous Integration and Deployment (CI/CD) pipelines enable early bug detection, reducing costs and ensuring smooth user experiences. Identifying defects in multi-layered microservices remains challenging. This report proposes a framework with failure attribution, baseline monitoring, automated revision reporting, and CI/CD alignment. The approach provides reliable, evidence-based debugging, minimizes effort, and leverages QA efficiency in complicated distributed systems using XPath-based object capture to validate the UI and historical payload baselines to test API interfaces.

BACKGROUND AND RATIONALE

Microservice-based architectures enhance scalability and flexibility but make quality assurance more difficult because they are distributed by nature and that services are interdependent. It is difficult to maintain consistency of data and easy interoperability of services that are developed independently and traditional testing is very difficult in isolating failures. Regression testing is necessary to ensure that the code remains stable, but it is not practical in agile and fast-moving cycles since it is a manual process. Automated regression testing also provides reliable feedback quickly, and minimizes risks [1]. The suggested framework improves the accuracy as the failures are attributed to UI, API, or cross-layer problems, and UI snapshots and API payload baseline baselines make the distinction between the intended and unintended changes clear.

SYSTEM ARCHITECTURE & PIPELINE DESIGN

Successful QA in microservices will involve a powerful architecture and pipeline, which will be based on automation, evidence-based analysis, and CI/CD integration. The centralized test automation system provides both API and UI testing. In the case of UI, object capture is based on XPath, which offers effective resistance to changes in the objects found in different builds, allowing a stable comparison of snapshots. In the case of APIs, the historical payload baselines ("golden copies") can be used to detect unwanted deviations. The pipeline is part of nightly regression and smoke testing, where it collects snapshots of UI and API payloads, compares them to baseline and produces detailed reports on changes [2]. This robotic system eliminates manual analysis and simplifies defect triage and also enhances reliability in a process of iterative development.

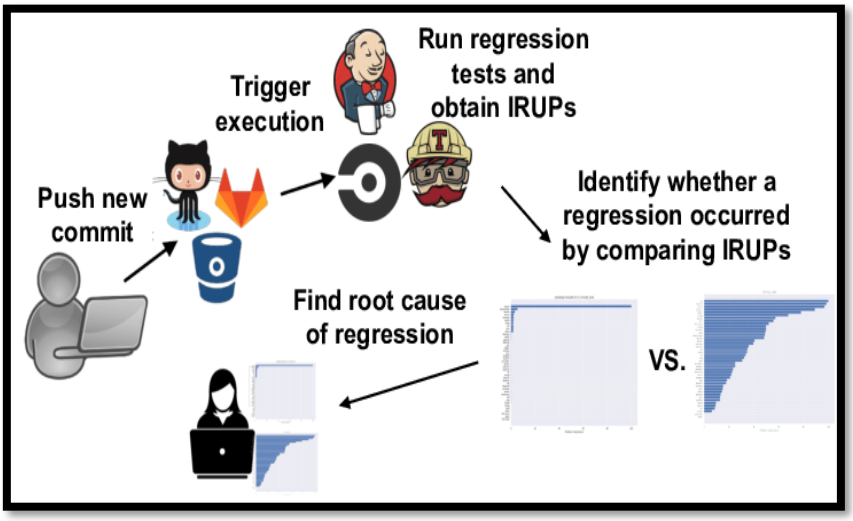

Fig 1: Automated QA Pipeline for Regression Detection

INTEGRATION & MAINTENANCE STRATEGY

The QA system should be implemented with smooth transition and maintenance. It integrates with CI/CD tools like Jenkins to run automated UI snapshots and API comparisons after each commit or build, ensuring continuous testing and fast feedback. A single repository stores golden baselines, updated to prevent false regressions. XPath locators should be robust, and regular audits of test suites and baselines ensure accuracy, stability, and long-term effectiveness [3].

SCALABILITY & OPTIMIZATION

The quality of software has to increase as software becomes more sophisticated. The main advantage of the identified procedure as automated regression testing process is scalability and optimization which will help to maintain the pace of the development process without losing the quality [4]. Another strategy of scaling which is important is the parallel implementation of tests. Modern CI/CD tools make it possible to run a large number of tests simultaneously, which can significantly reduce the overall time of the full process of running a full regression suite. This is especially valuable when a large-scale functional range is being tested. By spreading the tests out over several environments or containers, a short feedback loop to the developers can be maintained, despite increasing the number of tests.

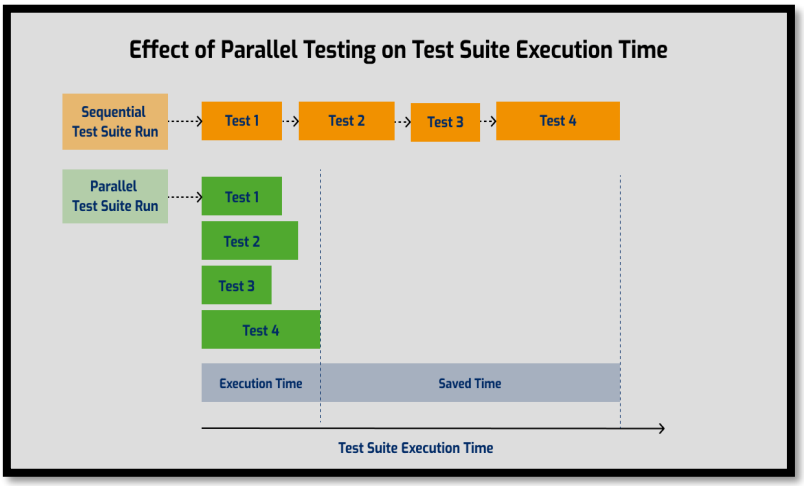

Fig 2: Optimizing Test Cycles with Parallel Execution

Another important factor is optimization of the test suites themselves. All the tests do not necessarily have to be executed with each code commit. It can introduce a tiered strategy to testing, with a smaller, critical set of "smoke tests" being run at any commit to give immediate feedback, and the entire regression suite then being periodically run, such as with nightly builds. Prioritization of the test cases TC The most important parts of the application are tested first with test case prioritization, ranking the tests according to the impact or risk they test or the level of risk the feature covered represents [5]. In the case of the API layer, comparisons of payloads can be performed by considering the structural integrity and critical points of data of the response, instead of a true comparison of the entire payload in a byte-by-byte manner. This not only can decrease the computational cost of the comparison process, but also render the tests less sensitive to non-meaningful variations, e.g. timestamps or dynamic-generated IDs.

REPORTING & MONITORING

Proper reporting and monitoring convert raw automated test data into actionable information to development and QA team, which is clear, concise, and timely information on the health of the application and impact of changes. The system should automatically generate a detailed revision change report after each run of each test. This report is used as a defect artefact report. It must point out clearly any differences that may occur between existing UI snapshots and API payloads and their corresponding baselines [6]. In the case of UI regressions, the report must provide a visual comparison on a side-by-side basis of the baseline and current snapshot with the dissimilarities to be presented visually. In the case of API regressions, the report needs to show a diff of the payload, whether it be of JSON or XML, and specifically include the fields that have been added, removed, or changed.

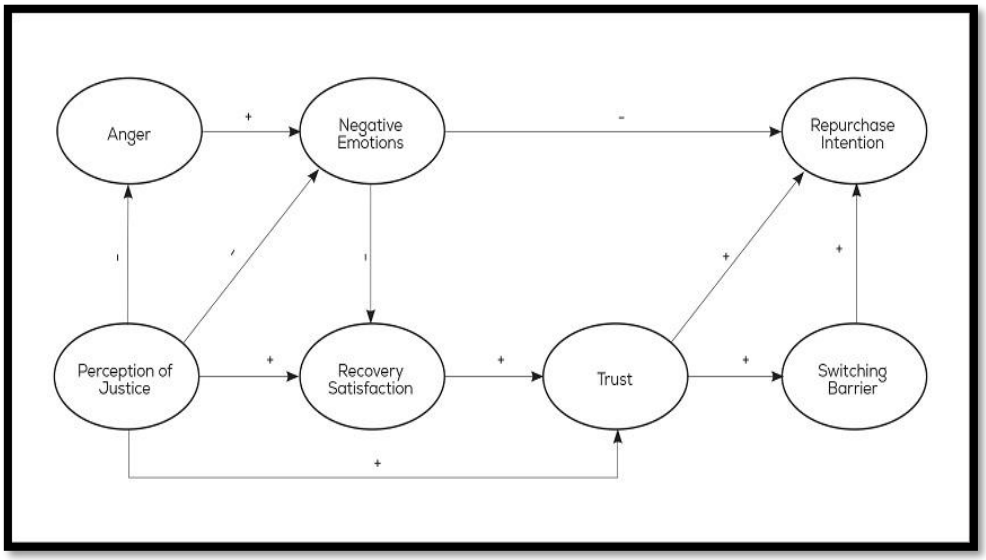

Fig 3: Evidence-Based Failure Attribution Model

This level of detail enables the QA team to be able to pinpoint the failure quickly and correctly. An example is the visual difference in the UI and an equal API payload is a sure sign of an issue in the UI. Conversely, in case the UI appears to be failing and an API payload is altered significantly, unintentionally, the change is an indication of an API-driven regression. Justifiable failure attribution ensures faults are assigned to the correct development team. Monitoring dashboards track test case outcomes and regression rates over time, providing stakeholders insight into software stability and reliability through the CI/CD pipeline.

CONCLUSION

The model is an evidence-based, strategic approach to QA of complex microservice systems instead of siloed testing. It enhances defects detection by failure attribution, baseline monitoring, automatic change reporting and CI/CD integration. APIs incorporate XPath-based UI validation and past payloads in the robust regression testing. Automation accelerates identification, eliminates human effort, and liberates QA departments to do advanced testing. The architecture and pipeline and maintenance plans optimize efficiency, reduce the cost of defects and offer more quality and reliable software to improve user satisfaction in the fast-paced development.

REFERENCES:

1. Fowler, M. (2000). Continuous integration. MartinFowler.com. https://martinfowler.com/articles/originalContinuousIntegration.html
2. Demircioğlu, E. D., & Doğru, A. H. (2022). API message-driven regression testing framework. Electronics,11(17), 2671. https://doi.org/10.3390/electronics11172671*
3. Leotta, M., Stocco, A., Ricca, F., & Tonella, P. (2016). ROBULA+: An algorithm for generating robust XPath locators for web testing. Journal of Software: Evolution and Process, 28(3), 177–204. https://tsigalko18.github.io/assets/pdf/2016-Leotta-JSEP.pdf
4. Koo, S., & Kim, J. (2023). Scalable test automation framework for large enterprise systems in CI/CD environments. Software Quality Journal, 31(4), 1099–1123. Springer. https://doi.org/10.1007/s11219-023-00680-y
5. Rothermel, G., Untch, R. H., Chu, C., & Harrold, M. J. (2001). Prioritizing test cases for regression testing. IEEE Transactions on Software Engineering, 27(10), 929–948. https://digitalcommons.unl.edu/cgi/viewcontent.cgi?article=1017&context=csearticles

6. Applitools. (2022, June 17). What is visual regression testing? Applitools Blog. https://applitools.com/blog/visual-regression-testing/
7. Internet Engineering Task Force (IETF). (2013). JavaScript Object Notation (JSON) Patch (RFC 6902). https://datatracker.ietf.org/doc/html/rfc6902