International Journal of Computer Trends and Technology
ISSN: 2231–2803 / https://doi.org/10.14445/22312803/IJCTT-V73I4P101

Volume 73 Issue 4, 1-4, April 2025
© 2025 Seventh Sense Research Group®

Original Article

Revolutionizing POS Terminal Testing with Python and CI/CD Automation

Avinash Swaminathan Vaidyanathan[1], Ajay Krishna Ramamurthy[2], Manoj Shankar Murugesan[3]

[1,2,3]*Independent Researcher, USA.*

[1]*Corresponding Author : avinash3788@gmail.com*

Received: 01 March 2025 Revised: 02 April 2025 Accepted: 16 April 2025 Published: 28 April 2025

Abstract - *In modern software development, ensuring the efficiency and reliability of Point of Sale (POS) terminal testing is essential for businesses. This paper introduces an automated testing framework using Python, designed to streamline the testing process for POS terminals. The A framework utilizes open-source libraries to interact with GUI elements and is deployed within VDI environments. Upon a commit in GitLab (CI/CD), a CI/CD pipeline job that launches a Selenium script to initiate the VDI will be executed. Based on the commit keyword, relevant test cases are executed automatically using Behave. Successful test case execution results in automatic code merging into production. This innovative approach significantly reduces manual effort and enhances the efficiency of POS terminal testing, providing a robust solution for continuous integration and delivery. By automating the testing process, businesses can achieve faster release cycles and maintain high-quality software standards.*

Keywords - *Automation framework, Continuous testing, DevOps, Gitlab, Pipelines.*

1. Introduction

Point-of-sale (POS) systems are critical for retail and service sectors, where transaction accuracy and system reliability directly impact customer satisfaction and revenue. Traditional testing methods often rely on manual execution and post-deployment corrections, which delay feedback and extend release cycles. In contrast, the current approach leverages Python-based automation within a CI/CD pipeline to test POS terminals pre-deployment. Much like a well-orchestrated assembly line that ensures every part is in sync, our framework integrates Python for GUI interaction. It behaves for behaviour-driven testing, triggering the appropriate test cases.

This paper details the framework's architecture, implementation, and impact on accelerating release cycles.

2. Literature Review

Automated testing in software development has been the subject of numerous studies, each highlighting various methodologies and tools. One area of focus is GUI-based automation, which utilizes tools like Selenium and PyAutoGUI. Selenium is known for its ability to test web applications across different browsers, making it common in many development environments. PyAutoGUI offers a solution for automating tasks on desktop applications, enabling comprehensive and versatile testing processes. These tools have streamlined testing workflows, identified user interface bugs, and enhanced the user experience of applications.

Another aspect of automated testing is Continuous Integration and Continuous Deployment (CI/CD). This methodology has been adopted in large-scale enterprise applications, where tools such as Jenkins, GitLab CI, and CircleCI play a role. CI/CD pipelines facilitate the automated testing and deployment of code changes, ensuring that every modification undergoes testing before being deployed. This practice minimizes human errors, accelerates development cycles, and maintains the application's integrity and functionality. Adopting CI/CD in enterprises underscores its reliability and scalability in modern software development.

An area that has received less attention is POS terminal testing within virtualized environments. Point of Sale (POS) terminals are critical in retail and hospitality industries, and their functionality is important. Virtualized environments like Citrix Virtual Desktop Infrastructure (VDI) offer solutions for POS terminal testing. These environments enable testers to replicate various configurations and scenarios without requiring physical hardware, providing a flexible and cost-effective testing approach. Few studies have specifically addressed this domain, highlighting a gap in the literature. Future research needs to explore methodologies and tools tailored for POS terminal testing within virtualized environments, aiming to improve coverage, efficiency, and overall testing outcomes.

While GUI-based automation and CI/CD practices are researched and adopted, POS terminal testing within virtualized environments remains underexplored. Addressing this gap presents an opportunity for future research to enhance automated testing methodologies, advancing the field and contributing to developing robust and reliable software applications.

3. Research Gap

Current automation approaches for POS terminals lack seamless integration within virtual desktop infrastructures and CI/CD pipelines. Existing methods either rely on manual intervention or are limited to non-virtualized environments, reducing their applicability in enterprise settings. Our proposed framework overcomes these limitations by fully automating test execution within a VDI and integrating with CI/CD for continuous testing and deployment.

4. Background

4.1. POS Architecture and Challenges

POS terminals typically operate within a layered architecture comprising hardware devices, middleware, and backend systems. The testing challenges include:

Hardware-Software Interactions: Verifying the seamless operation between touchscreens, card readers, and backend software.

- E2E Response time: Calculating End-to-End response time using the stopwatch approach is time-consuming.
- API response time: Identifying the correct APIs for each user action will be difficult as many APIs are involved.
- Access to components: Getting access to each component, like APIs, databases, and middleware, is challenging.

4.2. Existing Solutions and Their Gaps

Traditional testing methods lack the following:

- Automatic metric capture: Testers should manually record the response time for each user action using a stopwatch.
- Automated report: The test report must be prepared manually which is effort consuming.
- No integration: There is no integration with Gitlab for seamless test runs and JIRA to store the results.
- Build validation: Manually mark as pass / fail based on test results.

Table 1. Python framework vs Traditional method

Feature	Python framework	Stopwatch approach
Automatic report	Yes	No
Automatic test run for each change	Yes	No
3rd Party Integration	Yes	No

5. Materials and Methods

5.1. Framework Architecture and Design

The POS application is typically deployed across multiple Linux or Windows servers. Each server hosts instances of the POS application and might also run auxiliary services. The POS machines are connected virtually via the Tiger VNC server. Batch scripts are in place to automatically connect to a specific store's register.

- Python-Based Automation: Core testing routines are written in Python. These scripts drive the entire test workflow, orchestrating the interaction between various components.
- Selenium for GUI Testing: Selenium is used to simulate user interactions within the Citrix VDI. It automates the process of launching applications, interacting with GUI elements, and verifying expected behaviours.
- Behaviour-Driven Development (BDD) with Behave: Behave enables writing test scenarios in natural language. Test cases are triggered on specific commit keywords, ensuring only relevant tests are executed.

Sample behave feature test case:

```
@CustomGroup
Scenario outline: Apply coupon
Given: User login to the POS terminal using the
<username> and <password>
Then, the User navigates to the sale screen
Then, the user scans an UPC
Then, the User clicks the total
Then, the user applies the coupon
Then, the User pays the remaining amount via cash
Then, the User completes the transaction
Examples:
    user1 | password 1 |
```

5.2. Implementation Steps

- GitLab setup: Set up the CI/CD pipeline to enable continuous testing and build validation. GitLab CI/CD configurations are tailored to detect specific commit keywords, triggering only relevant test suites.
- Report integration: To store the test results in a centralized server cloud, preferably using 3rd party tools like JIRA, etc.
- Post Hook: Develop an API to execute the right test cases based on the commit made.
- Let's trace the End-to-end flow as below,

  Whenever a commit is made in the GitLab repository, the post hook API will trigger an automated python selenium script to launch the VDI in browser.

  Once the VDI is logged in, the startup script to launch the Tiger VNC to connect to the POS application will be triggered.

Then, the startup script to launch the Tiger VNC to connect to the POS application will be triggered automatically. Then, the relevant test cases will be executed automatically, and the results will be updated in the JIRA ticket.

The results from JIRA will then be sent to GitLab to mark the build status as Pass / Fail. If pass then, then the build will be deployed to production automatically.

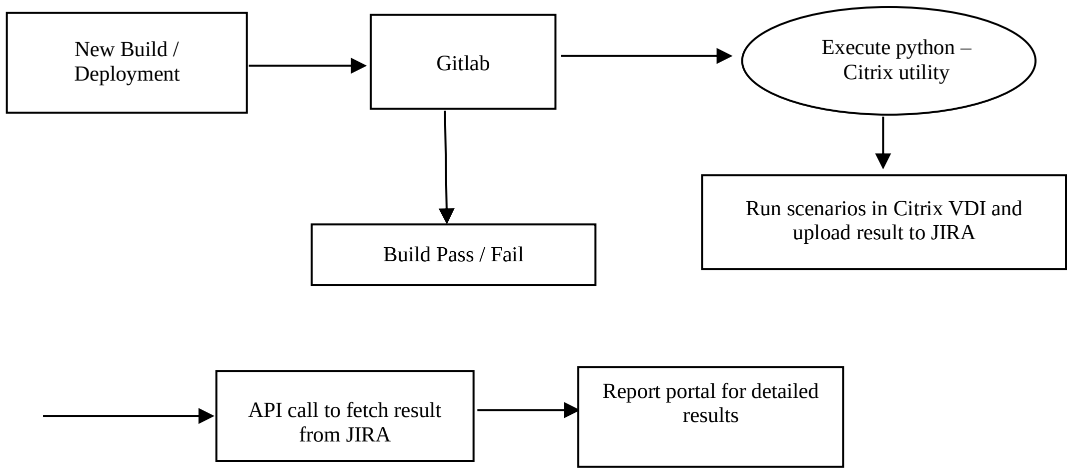
New Build / Deployment
Gitlab
Execute python – Citrix utility
Run scenarios in Citrix VDI and upload result to JIRA
Build Pass / Fail
API call to fetch result from JIRA
Report portal for detailed results

Fig. 1 Flow diagram

6. Results and Discussion

6.1. Performance Analysis

The automated framework was evaluated in a controlled retail environment with multiple POS terminals simulated in the Citrix VDI. Key observations include:

Test Execution Time: Average test runs were reduced by approximately 30% compared to manual testing.

Error Detection: Early defect detection increased by 40%, thanks to real-time validation during the CI/CD process.

Test Coverage: Test coverage increased by 30% compared to manual testing.

Table 2. Performance comparison

Aspects	Traditional	Python framework
Test Execution time	30 Minutes	9 Minutes
Error Detection	Only during QA	Immediately
Test Coverage	30%	60%

6.2. Comparative Analysis with Traditional Methods

Traditional testing approaches often rely on scheduled batch tests, resulting in delayed feedback and prolonged bug resolution. In contrast, our automated framework provides:

Continuous Feedback: Instant validation upon code commits.

Higher Accuracy: Automation reduces human error and variability.

Faster Release Cycles: Immediate merging of validated code leads to accelerated deployment schedules.

6.3. Challenges and Limitations

While the framework demonstrates significant benefits, several challenges were encountered:

Initial Setup Complexity: Configuring the Citrix VDI and integrating various tools required substantial upfront effort.

Learning Curve: Developers needed to adapt to the BDD approach with Behave.

Scalability Considerations: Although effective for small-to-medium deployments, further optimizations are necessary for extremely large retail networks.

7. Experimental Setup

Our test environment consists of:

7.1. Hardware

POS terminal hardware with a Linux-based OS.

Remote Server: A Windows server is required to run the Python framework and to launch the POS application via the TigerVNC server. The server should run with maximum uptime of no less than 99.999%

7.2. Software

Selenium, Python with the below dependencies, and GitLab CI/CD.

- Behave
- pygetwindow
- opencv-python
- pyautogui
- pywinauto
- easygui
- pyodbc
- beautifulsoup4
- selenium

7.3. Testing Procedure

A GitLab pipeline triggers test execution upon a commit, launching a Selenium script to access the VDI, executing relevant test cases, and reporting results.

7.4. Validation

Test results are compared against expected outputs, and failed tests prevent automatic merging.

8. Case Study: Implementation in a Retail Environment

8.1. System Configuration

A retail chain with 60 POS terminals was selected as a pilot site. Each terminal was virtualized using Citrix VDI, and the testing framework was integrated into the organization's GitLab repository.

8.2. Observations and Outcomes

- Transaction Processing: The automated tests simulated real-world transactions, identifying configuration errors and software bugs.
- Cycle Time Reduction: Deployment cycle times were reduced from days to hours.
- Error Resolution: The time to resolution for critical errors dropped from 45 minutes to under 10 minutes on average.

8.3. Lessons Learned

The case study highlighted the importance of:

- Robust Environment Replication: Ensuring the test environment closely mirrors production conditions.
  Clear Test Criteria: Defining precise commit keywords to trigger relevant test cases.
  Continuous Monitoring: Using dashboards to track test performance and quickly address anomalies.

9. Future Work

9.1. Enhancements in Test Automation

Future efforts will focus on:

Expanding test libraries to cover more POS scenarios. Incorporating real-time reporting features to enhance feedback loops.

9.2. Integration with AI and Machine Learning

Leveraging GenAI could enable:

Predictive maintenance by identifying recurring issues.

Adaptive testing that dynamically adjusts test scenarios based on historical data patterns.

9.3. Expanding to Additional Environments

Extending the framework to support mobile POS systems and IoT-based devices. Enhancing cross-platform compatibility to accommodate various operating systems beyond Citrix VDI.

10. Conclusion

The introduction of a Python-based automated testing framework for POS terminals marks a significant advancement in pre-deployment testing. Integrating with a CI/CD pipeline and utilizing tools such as Selenium and Behave reduces manual effort, accelerates release cycles, and maintains high-quality software standards. This approach is particularly beneficial in dynamic environments where rapid iteration and robust testing are paramount.

Acknowledgments

All the 3 authors contributed equally to this work.

References

[1] Python Documentation, Python Software Foundation, 2025. [Online]. Available: https://www.python.org/doc/

[2] SeleniumHQ, Selenium WebDriver, 2025. [Online]. Available: https://www.selenium.dev/documentation/en/

[3] Behavior Driven Development in Python, Behave Documentation, 2025. [Online]. Available: https://behave.readthedocs.io/

[4] What is CI/CD?, CI/CD, 2025. [Online]. Available: https://about.gitlab.com/topics/ci-cd/