SSRG International Journal of Computer Science and Engineering
ISSN: 2348–8387 / https://doi.org/10.14445/23488387/IJCSE-V12I19P102

Volume 12 Issue 9, 8-11, September 2025

Original Article

In-Sprint Automation: A Much Needed Cultural Shift to Accommodate Generative AI Storm in Quality Engineering Space

Nagmani Lnu

Director of Quality Engineer at a FinTech Company, San Antonio, USA.

Corresponding Author : nagmanijobs@gmail.com

Received: 17 July 2025 Revised: 18 August 2025 Accepted: 05 September 2025 Published: 29 September 2025

Abstract *- As Gen AI grows, the demand to include Artificial Intelligence in the software testing life cycle will increase to achieve faster release cycles. Quality engineers (QE) will face even more pressure to deliver automated tests within the same sprint as development. This paper presents a practical framework and cultural transformation adopted by a mid-size FinTech enterprise that enabled 90% in-sprint automation coverage. The approach redefines the QE role, leverages method-level reusability, and introduces placeholder scripting to handle partial requirements early in the sprint. Workshops, proofs of concept, and organizational-level rollouts played key roles in scaling this model. Our case study demonstrates how cultural shifts and strategic reuse can bridge the gap between agile development and test automation. This methodology aligns well with the DevOps principle of "shift-left" testing, where test activities are integrated earlier in the lifecycle to reduce feedback cycles and improve quality.*

Keywords *- Generative AI, In-Sprint Automation, Selenium, Shift-left Testing, Test Automation.*

1. Introduction

Ever since Agile development began, companies have recognized the need for continuous integration and continuous delivery, which have become central themes in their development processes. Software testing is an important part of the software development life cycle. Companies dedicate a significant amount of development time to testing code, and the ongoing demand for automation is ever-present to accelerate the overall development process.

The traditional QE operation followed a three-phase model:

1. Sprint Planning: QEs write test cases.
2. Sprint Execution: QE executes test cases.
3. Post-sprint: Automation engineers automate manual cases or partially generate them through Artificial Intelligence (AI)

This model has created multiple inefficiencies:

- Automation lagged development.
- Duplicate effort is made using manual steps and then automated steps.
- Additional QE headcount is needed to close gaps.
- Stories developed late in the sprint go unautomated for multiple cycles.

In this model, automation is treated as an afterthought rather than a continuous practice. Manual testing is prioritized as the primary deliverable, while automation is relegated to a secondary activity, addressed only if time allows. This is often challenged, especially in Agile development practices, as Agile facilitates rapid, iterative delivery of software but faces the headwind of automation not being completed within the same sprint. According to the Capgemini World Quality Report 2023 [1], 70–80% of respondents reported that their QE teams lack sufficient time for automation. While Generative AI may reduce this percentage, scalable in-sprint automation will remain challenging [2] unless tests generated by Gen AI are managed properly for reusability. An approach must be established within the Quality Engineering team to utilize these tests for *in-sprint automation*, which requires both technical and cultural changes within the organization. This paper outlines an approach to transition to a model where automation occurs within the same sprint without sacrificing quality or velocity and covers the technical and behavioral changes an organization can implement to enable in-sprint automation.

2. Literature Review

In-sprint automation has become a critical requirement from the outset of agile development, and the industry is increasingly looking to leverage Generative AI to address this

need. In 2024, Joshua Moses published an article on using Generative AI for code generation [3]. In 2023, V. Shobha Rani, Dr A. Ramesh Babu, K. Deepthi, and Vallem Ranadheer Reddy published an article discussing the benefits, challenges, and best practices of shift-left testing in DevOps [4]. In 2024, Thamiziniyz Natrajan and Shanmugavadivu Pichai published an article on a Behavior Driven Development and metrics framework to enhance agile practices [5]. None of these papers addresses the approach and process, including cultural change and method reusability, required to adopt in-sprint automation at scale for the enterprise. This paper stands out in that respect.

3. Solution Approach

To address in-sprint automation challenges, the following exercises were conducted to resolve technical and cultural issues:

- Method Reusability Awareness: Most UI or API flows share common interactions (e.g., login, navigation, form submission). The study showed that 60–80% of the steps in manual test cases had already been automated in previous sprints. QEs were empowered to leverage these reusable methods at the start of each sprint, reducing the amount of code they needed to write from scratch and ensuring consistency across test suites. The automation framework was further enhanced with tagging and metadata, making it easier and faster to locate reusable components.
- Deferring automation to the next sprint for stories arriving late because of development delay: Stories that arrived late in a sprint were scheduled for automation in the following sprint. However, shared steps for these stories were still pre-automated using reusable methods, ensuring that some level of automation coverage was consistently maintained. Additionally, deferred automations were tracked independently, and QE metrics incorporated "delayed automation coverage" to monitor any gaps.
- In-Sprint Test Reviews and Automation Syncs: To enforce culture, mid-sprint checkpoints were introduced, during which QEs presented completed automation scripts in brief sync meetings. This practice fostered alignment with developers, encouraged code reuse, and helped identify blockers early. Additionally, peer reviews became mandatory for every new script to uphold code quality and framework standards.

4. Implementation Approach

A pilot initiative was launched within a single agile team and tracked over three sprints to measure the following:

- Reuse rate of existing automation methods
- Time spent on manual test design versus automation
- Regression readiness by sprint close
- Number of scripts completed within sprint boundaries

One of the primary obstacles was shifting mindsets, as engineers were accustomed to handling manual and automation workflows separately. Several workshops were held to demonstrate the following:

- How to identify reusable methods
- Writing flexible scripts with placeholder support
- Git branching strategies to avoid conflicts with incomplete features
- How to convert acceptance criteria into BDD-style tests that can be automated early

Leadership support proved essential. Engineering managers established in-sprint automation as a key performance indicator (KPI) and formally recognized early adopters. These cultural shifts had a notable impact. The results demonstrated up to 85% automation readiness within the sprint. Manual testing efforts decreased, and regression tests were executed on the same day as feature delivery. After the initial team's success, Sprint Automation was scaled to five agile teams throughout the organization. Automation repositories were modularized, method libraries were standardized, and peer-review processes were implemented. Additionally, a central QE guild was formed to share best practices and monitor adoption metrics.

5. Test Cases and Code Examples

As part of the workshop exercise, the following examples were automated

5.1. Test Case 1

Loan Payment Verification: This test case verifies that after a user logs in, navigates to the loan page, selects a loan, makes a payment, and logs back in, the payment is reflected correctly. As part of the working session, the task was to analyze the available pages and reusable methods, and to create a placeholder method for the new functionality that was still under development, as shown in Table 1.

Table 1. Test Case 1

Step	Action	Reusable?	Notes
1	Login	Yes	Reused login method
2	Select loan	Yes	Reused the loan selection method
3	Pay the loan in full.	No	New logic to pay off the loan went as a placeholder.
4	Check loan payment status.	No	New logic to verify payment status went as a placeholder.

Once the flow was identified, it was time to create the automation. The Page Object Model (POM) and Test class code illustrated in Figures 1 and 2, respectively, serve as an example to explain the concept. The actual written code cannot be shared due to security concerns.

```
public class LoanPage {
    WebDriver driver;

    // Constructor
    public LoanPage(WebDriver driver) {
        this.driver = driver;
    }

    // Method to select a loan
    public void selectLoan(String loanId) {
        driver.findElement(By.id("loanSelection")).sendKeys(loanId);
        driver.findElement(By.id("selectLoanButton")).click();
    }

    // Method to pay off the loan in full
    public void payOffLoan() {
        driver.findElement(By.id("payLoanButton")).click();
        driver.findElement(By.id("confirmPaymentButton")).click();
    }

    // Method to verify loan payment status
    public boolean isPaymentSuccessful() {
        return driver.findElement(By.id("paymentStatus")).getText().contains("Paid");
    }
}
```

Fig. 1 POM Example to Automate the Loan page

```
public class LoanPaymentTest {
    WebDriver driver;
    LoginPage loginPage;
    LoanPage loanPage;

    @Before
    public void setUp() {
        driver = new ChromeDriver();
        loginPage = new LoginPage(driver);
        loanPage = new LoanPage(driver);
    }

    @Test
    public void testLoanPaymentVerification() {
        loginPage.login("testuser", "password123");
        loanPage.selectLoan("loan123");
        loanPage.payOffLoan();
        Assert.assertTrue("Loan payment was not successful", loanPage.isPaymentSuccessful());
    }

    @After
    public void tearDown() {
        driver.quit();
    }
}
```

Fig. 2 Test Class Example code

5.2. Test Case 2

Loan Visibility After Payment: The test case was to ensure that when a user logs in, pays off a loan, and logs back in, the paid loan is no longer displayed in their loan list.

The reusability analysis is shown below in Table 2.

Table 2. Test Case 2

Step	Action	Reusable?	Notes
1	Login	Yes	Reused login method
2	Select loan	Yes	Reused the loan selection method
3	Pay the loan in full.	No	New logic to pay off the loan
4	Log out and log back in.	Yes	Reused login/logout method
5	Check if the loan is displayed	No	New logic to verify loan visibility post-payment

The code in Figure 3 below serves as an example to demonstrate the automation process.

```
public class LoanVisibilityTest {
    WebDriver driver;
    LoginPage loginPage;
    LoanPage loanPage;

    @Before
    public void setUp() {
        driver = new ChromeDriver();
        loginPage = new LoginPage(driver);
        loanPage = new LoanPage(driver);
    }

    @Test
    public void testLoanVisibilityAfterPayment() {
        loginPage.login("testuser", "password123");
        loanPage.selectLoan("loan123");
        loanPage.payOffLoan();
        loanPage.logOut();
        loginPage.login("testuser", "password123");
        boolean loanVisible = loanPage.isLoanVisible("loan123");
        Assert.assertFalse("Loan should not be visible after payment", loanVisible);
    }

    @After
    public void tearDown() {
        driver.quit();
    }
}
```

Fig. 3 Test Class for Test Case 2 Example code

6. Results

The new model achieved:

- 90% automation completion within the sprint
- 70% reduction in automation creation effort by the centralized team
- Enable a centralized team for other enabler activities
- Higher release confidence
- Reduced test debt and maintenance overhead

7. Conclusion

While AI can help generate code snippets, it alone will not enable in-sprint automation, as in-sprint automation is not just a technical upgrade—it is a cultural transformation. This study emphasizes the importance of early QE involvement, smart reuse, and flexible scripting. By shifting automation activities earlier and empowering QEs to take ownership of test development, significant efficiency and quality gains can be achieved. Future directions include integrating generative AI for dynamic placeholder filling, NLP-based test case parsing, and self-healing scripts to reduce maintenance.

References

[1] World Quality Report 2023-24, Capgemini. [Online]. Available: www.capgemini.com/insights/research-library/world-quality-report-2023-24/

[2] Rahul Jain, Overcome the In-Sprint Automation Challenges with Test Automation [2025], LambdaTest, 2024. [Online]. Available: https://www.lambdatest.com/blog/top-in-sprint-test-automation-challenges/

[3] Joshua Moses, Accelerating Agile Sprints with Generative AI for Code Automation, 2024. [Google Scholar]

[4] V. Shobha Rani et al., "Shift-Left Testing in DevOps: A Study of Benefits, Challenges, and Best Practices," *2023 2nd International Conference on Automation, Computing and Renewable Systems (ICACRS)*, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[5] Thamizhiniyan Natarajan, and Shanmugavadivu Pichai, "Behaviour-driven Development and Metrics Framework for Enhanced Agile Practices in Scrum Teams," *Information and Software Technology*, vol. 170, 2024. [CrossRef] [Google Scholar] [Publisher Link]