Test Automation Strategies in a Continuous Delivery Ecosystem

Testing organizations must always balance quality with continuous delivery. By adopting intelligent automation tools and techniques, they can simultaneously maintain agility and enhance development efficiencies.

Executive Summary

To keep pace in the digital era, enterprises must modernize how they build applications in order to meet users’ ever-rising business requirements and manage ever-faster releases of new features and functionalities. To make this happen, continuous integration (CI) techniques can be applied to ensure hoped-for results are delivered by rapid development models such as Agile and extreme programming (XP).

While automation is the basis of CI, test automation – in a CI construct – is fraught with challenges, and is easier said than done. Each code integration is verified by an automated build followed by an automated “smoke” test, allowing teams to detect problems early in the lifecycle. It is important to understand and overcome these challenges and find a way to automate tests by applying CI techniques to enable seamless and continuous delivery (CD).

This white paper highlights the role of CI in CD, identifying the challenges of test automation in a CI environment and exploring various automation techniques that improve agility in continuous testing.

Role of Continuous Integration in Continuous Delivery

With mounting business demands to accelerate time to market, we see more teams migrating from traditional development models to continuous delivery based on Agile development methods. Through this approach, software is developed in short cycles, thereby ensuring reliability for timely releases. This results in building, testing and releasing the software faster and more frequently. The approach has proven to reduce cost, time and the risk of delivering critical changes to production, thereby allowing incremental updates to the production system.

CD is driven by CI. In CI, the code is logged-in several times a day and then re-compiled, generating multiple QA feedback loops. Thus, it is necessary for development teams to collaborate and make frequent deployments, which mandates greater levels of process automation. Also, to successfully apply CI, automation of the build and deployment process is critical; this ensures a self-testing build and makes the entire process transparent and agile. Figure 1 (see next page) depicts the involvement of automation in CD.
Continuous Integration: The Foundation for Continuous Delivery

<table>
<thead>
<tr>
<th>Automated Processes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dev/Test Provision</td>
</tr>
<tr>
<td>Test</td>
</tr>
<tr>
<td>Release &amp; Roll-back</td>
</tr>
<tr>
<td>Monitoring &amp; Incident Resolution</td>
</tr>
<tr>
<td>Plan</td>
</tr>
<tr>
<td>Code</td>
</tr>
<tr>
<td>Build</td>
</tr>
<tr>
<td>QA Deploy</td>
</tr>
<tr>
<td>Continuous Operations</td>
</tr>
<tr>
<td>Continuous Delivery</td>
</tr>
<tr>
<td>Project promotion</td>
</tr>
<tr>
<td>IT Services management</td>
</tr>
<tr>
<td>Agile Development</td>
</tr>
</tbody>
</table>

Figure 1

Challenges in Automating QA in Continuous Delivery

As specifications evolve and must be continuously integrated into a product’s final code, the time and scope for a release in the development cycle are limited. Moreover, frequent builds may also decrease the stability of the code. Hence, it is crucial to address QA in a more robust manner. Often, during Agile development projects, the functionalities and UI change across multiple Sprints; hence, automating QA is far from an easy task. In our engagements, we have experienced the following test automation challenges in Agile delivery projects:

- **Ensuring test coverage**: As code is integrated continuously, chances are that critical tests for a particular requirement could be missed. Also, unanticipated code changes could lead to limited test coverage during automation.

- **Costlier defect fixes**: Where test coverage is not complete, defects that belong to an earlier Sprint are detected much later in the development cycle. Once the application is in production, these defects prove to be significantly costly to fix.

- **Compromise of user performance**: As software features are added incrementally, more lines of code are generated, leading to performance

Quick Take

**Reducing Costs by 43% for a Leading U.S. Based Provider of Group Disability Insurance**

After adopting an Agile development methodology, a large U.S.-based provider of group disability benefits lacked standard frameworks, processes and QA disciplines for test automation. There was a high dependency on manual kick-off of automation suites and resources. There were over 200 application test cases, but there was no way to determine an optimal suite for a particular package.

We helped this client automate early by leveraging a customized componentization test framework that enabled its development and QA teams to adapt to frequent application changes in its newly-minted Agile development environment. Our QA teams also built a robust regression suite over multiple Sprints, thereby helping the client achieve a 72% improvement in test coverage and reduce testing costs by 43% over successive Sprints. For the business, these advancements in the software delivery pipeline enabled shorter time to market for new product features, with fewer defects and reduced downtime.
Early Test Automation Model for Continuous Integration

Figure 2

Quick Take

Delivering a 72% Reduction in Testing Efforts to a Leading Multinational Bank

We undertook a large-scale database migration for a multinational bank with a focus on investment banking, asset management, private banking, private wealth management, treasuries and securities services. This required that the validation of functional flows residing in over 25,000 database objects be deferred to functional testing. These database objects constituted approximately 60% of functional behavior.

We automated the database validation of stored procedures and the generation of input parameters. These automated processes accounted for 60% of functional flows before UI readiness. The net result: a 72% reduction in testing efforts.
issues, especially if development teams are not aware of the impact that code changes could have on end-user performance.

As developers work in collaboration with testers, it is important to ensure that persistent, rapid testing and fixes are performed continuously.

Thus, despite the aforementioned challenges, CI automation will play a critical role in expediting the delivery pipeline by completely eliminating wait time, and it sets the foundation for CD. These challenges all need to be duly addressed to ensure seamless CD.

**Automation Technique #1: Achieve Continuous Regression**

QA organizations must keep pace with evolving development methodologies. As automation becomes inevitable in an Agile development world, it is important to automate early for CI and CD. Through early automation, the building blocks of automation are applied much earlier in the Agile development cycle. Early automation enables efficient and automation-friendly test designs, feeding into an efficient functional cycle. As a result, continuous regression that exercises iteratively-built code results in a stronger automation foundation. Figure 2 (previous page) depicts our approach to early automation.

Through early automation, functional testing is fast-tracked. This approach not only reduces the efforts of repetitive functional tests prerequisites, but expedites overall functional test execution. Early automation helps teams to move data-driven functional tests directly to automation (see Quick Take, page 2).

**Automation Technique #2: Think Before UI**

In CI, the UI layer is built over successive iterations. However, a complete user interface is not available for UI-based automation. Therefore, QA organizations must consider unconventional automation models to automate early. Applications with multitier architectures built on Web services/APIs, messaging middleware, and databases pose an opportunity to validate business rules, logic and functionality at intermediate layers as well as on back-end systems. They communicate with each other through messaging protocols, but most of the critical business logic resides in the database in the form of various database objects.

Looking beyond UI creates opportunities to utilize the communication layer to simulate any upstream application behavior, thus reducing the dependency on upstream applications. Similarly, any downstream check on the data flow can be

**Early automation enables efficient and automation-friendly test designs, feeding into an efficient functional cycle.**

---

**Quick Take**

**QA Costs Halved for a Leading U.S. Healthcare Provider**

This healthcare client planned weekly release cycles to major applications, which involved execution of hundreds of functional tests. Time constraints for product releases meant that these tests were performed in parallel to development. Due to lack of visibility on the test cases that were impacted by code changes, these tests were executed several times over the course of a release. As a result, the same requirements were tested more than once while a few requirements were not tested at all, impacting coverage and cost.

To alleviate this, we deployed an analytics-based code coverage framework, which identified the critical tests and reduced testing cycles from 239 to 66 and reduced overall QA costs by 50%.
performed by verifying the status in downstream databases or validating the returning messages. We have built a few utilities around message-based validation and database object validation, such as Quetest and Object Test Pro, which can be used to enable component automation. (see Quick Take, previous page).

**Automation Technique #3: Optimize Testing at Code Level**
Conventional requirements-based testing of code changes tends to result in considerable over-testing without the guarantee of complete coverage. The impact of missed tests on coverage and quality is significant in CI. This typically results in additional QA cost and extended testing timelines, that in turn impact project success. We developed a solution that automates the impact analysis of code changes to affected test cases by scanning the application build and by automatically mapping code components from the build to test cases. This ensures optimized testing while maximizing test coverage. (see Quick Take, previous page).

**Automation Technique #4: Minimize Manual Intervention**
As development and testing increasingly run in parallel, stakeholders need better collaboration. Development teams achieve greater collaboration if they are able to select the right tools, frameworks and methods to automate CI. The selection of the right tool could help in improved deployment success and minimize the broken builds.

### Deployment Time Slashed 60% for a Large North American Motor Club

This client’s QA teams were challenged to keep up with aggressive multi-state roll-out schedules of the company’s policy administration system. Frequent builds across multiple environments, each having different configurations, required more efficient ways of certifying the builds so that subsequent testing services of various kinds can be performed across environments.

**Development teams achieve greater collaboration if they are able to select the right tools, frameworks and methods to automate CI.**

This is vital, especially when organizations work on multiple code changes during a single day. CI servers pick up the daily commits and create automated builds, which are then deployed daily to the build environment. Key stakeholders review the findings on a daily basis and address those before the next code log. We created frameworks across CI functions, such as Craft, SPRITZ and analytics-based solutions that provide a snapshot of the status at any given time.

### Looking Forward
As modern development techniques based on Lean principles emerge, QA teams have to keep pace with the agility of development teams. While automation is inevitable in the building and deployment of rapid development models, QA automation becomes increasingly challenging.

A single solution is not possible. However, by adopting an intelligent automation technique that automates an application at every stage of its journey – from a code to a component to the UI – QA organizations can offset the challenges that are inherent in conventional CI automation techniques.
Footnote


About the Authors

Sriram Sitaraman is a Technology Leader of the Automation Center of Excellence (NA) within Cognizant’s Quality Engineering and Assurance business unit. He has 14-plus years of experience in IT, with a focus on functional and automation testing. Sriram has hands-on experience with multiple industry-leading automation and test management tools. Specializing in development and implementation of automation frameworks and solutions for the Agile methodology, he is an experienced automation architect who focuses on automation strategy, design and architecture. Sriram is an innovative professional with a proven ability to identify, analyze and solve problems to increase customer satisfaction. He received his master’s degree in computer science from University of Madras. Sriram can be reached at Sriram.Sitaraman@cognizant.com.

Rameshkumar Bar is a Senior Automation Architect with Cognizant’s Technology Center of Excellence within its Quality Engineering and Assurance business unit. He has 13 years of experience in the information technology field, of which the last five years have been with Cognizant. Ramesh has extensive experience in developing Web applications on the .NET platform, and building test automation frameworks using commercial and open source tools. His areas of specialization include test automation, continuous integration and test data management. He received an engineering degree in information technology from Saurashtra University, India. Ramesh can be reached at Rameshkumar.Bar@cognizant.com.