coveros

Coveros Implements Test Automation at PECOS 2.0

CASE STUDY

CHALLENGES

- QA team members didn't possess necessary automation skills
- Test efforts lacked a structure for automation
- Testing was isolated to individual QA members
- Testers documented and scripted tests in their own separate repository

SOLUTIONS

- Automated testing was prioritized and teams got training
- A front-end testing framework was implemented that integrated with the CI/CD pipeline
- Test cases were assembled into containers for pipeline builds
- Microservice functionalities were compiled and published to one release repository

CASE STUDY

Coveros Implements Test Automation at PECOS 2.0

The Centers for Medicare & Medicaid Services (CMS) sought to revamp the Medicare enrollment process by developing a replacement Provider Enrollment, Chain and Ownership System (PECOS). The goals of the system were to reduce provider burden and improve operational efficiency while strengthening program integrity. PECOS 2.0 was a ground-up rebuild of the application system and enrollment process, leveraging modern technology to deliver simplified processing workflows for users, greater interoperability among programs, and increased transparency by offering an enterprise platform for all provider enrollments across Medicare, Medicaid, and emerging programs.

In 2017 Solutions By Design II (SBD) was awarded the program and teamed with Coveros to implement the solution. The project was implemented using an agile and DevOps approach, with a half-dozen agile development teams building, testing, and deploying the microservices that make up PECOS 2.0 on a continuous basis. Agile testing with a heavy emphasis on automation was necessary to ensure high-quality software releases.

While development teams put a strong emphasis on unit testing to achieve high levels of code coverage, functional testing was largely neglected during the early phases of the project. Like many new development projects, siloed testing and development led to a large amount of testing technical debt, as well as the ownership of quality being left on the QA team late in the lifecycle. Furthermore, the team never adopted a test-first approach, so testing was always pushed off until later, causing a ripple effect in delays. Once testing did occur, initial testing was all manual and exploratory, and test cases weren't fully documented. Tests were hard to reproduce in a manual or automated fashion, no requirements for test traceability existed, and regression and release risk were difficult to define. Testing also couldn't be done in sprint because there was no shared understanding of the application between dev and QA.

CHALLENGES

- Automated testing was initially undervalued, so some QA members were hired without automation skills and needed training and a structural framework
- Initial testing efforts suffered from a lack of emphasis on automated testing and test architecture, with no framework to organize, design, and implement tests that could integrate properly with the development code base and the CI/CD pipeline
- Developers viewed testing as a separate problem for the QA silo, and the definition of done for stories did not include automated testing, or even formal documented verification procedures for features
- Automated tests were solely owned and used by the QA team, so upkeep was a problem, pipeline integration was often an afterthought, and other teams

"Adapting Selenified framework into PECOS 2.0 helped immensely in kick-starting the team's UI automation efforts. It was easy to use, and the test case reports generated were highly valuable. Having the Coveros test solution lead available for answers and to conduct working sessions boosted the team's confidence in adapting to this new framework."

—Sukanya Somasundaram, QA team lead on PECOS 2.0

didn't know how to run tests (or even if they existed)

- Testers documented and scripted tests in their own separate repository area, leading to tests drifting out of sync with the actual code they were running against, causing difficulty in finding and retrieving the right tests

SOLUTION

Coveros engaged with the existing test engineers to develop a plan of action for improving the testing cycle and efforts made by the QA team.

The first order of business was to introduce a front-end testing framework to support the application under development. The Selenified test framework was chosen due to its open source nature and ease of use for both web and API testing. A few UI and API tests were written as examples, after which the QA team was trained on the tooling and implementation. The team also started an effort to convert existing tests written using Cucumber and Selenium to Selenified. This was an easy way to introduce writing tests using Selenified, and it could use the tests previously written by the QA team. Working training sessions were held individually and as a team multiple times a week to get testers up to speed.

The next step was to implement a structure and method for storing and updating tests. Due to the nature of the application with its multiple microservices, the application existed in multiple Git repositories. Each repository contained not just the microservice code, but also the automated tests associated with the UI or API. This ensured that the tests stayed in sync with the code as it evolved. As code was updated on branches, the tests would get updated at the same time.
In order to facilitate simpler testing, both locally and in the pipeline, the Selenified tests got assembled into Docker "test containers" for each service during pipeline builds. The Dockerized test containers made it easy to package and run tests that matched the version of each microservice. On a successful build—meaning all tests passed—the Docker container was pushed to the artifact repository, Nexus, so that all users could download and execute these tests with a single command.

The DevOps pipeline was set up so that the tests were executed in the pipeline any time a GitHub pull request was made. Regardless of whether the tests passed or failed, a report was generated in Jenkins that outlined each test, its status, and individual test steps, to make debugging simple. If there were any test failures, the testers would be notified so that they could make the necessary changes. This rapid feedback also allowed the testers to become aware of any major code changes and modify the tests as needed. This drove further communication between the testers and the development teams.

In order to eliminate any test and framework page duplication, the team made heavy use of the Page Object Model (POM) and DataProviders, following common test automation design patterns. In order to speed up

"Coveros was able to come in, suggest appropriate tooling, and train our staff to write functional testing in a short period of time. This common tooling allowed us to increase our test automation coverage and improve confidence in the quality of the software."

—Jay Bercher, SBD deputy program manager for PECOS 2.0

test execution, steps of each test that were not dependent on the UI were done directly through the API. This pattern allowed automated test cases to focus on what mattered for validation, rather than getting hung up in other areas.

Because there were heavy dependencies between the microservices, tests often needed to reference functionality from another service; for example, a dashboard test needs authentication functionality. To further reduce duplication, the functionalities from each service's Gradle build were compiled, stored, and shared in Nexus. This easily allowed all services to reuse the code as a common library. Each pull request had the common step libraries pushed to a snapshot repository in Nexus so that initial common steps could be shared.

Once tests passed in the pipeline and were merged into development, these common libraries were published to a release repository in Nexus. A semantic versioning model was followed with these libraries in order to keep breaking changes from impacting existing dependent tests.

TECHNOLOGY SOLUTIONS

- Selenified testing framework for front-end and API tests
- JUnit and Jest for unit tests
- Spring Boot Java REST services
- React JS web UI
- Docker containers running in OpenShift Kubernetes Distribution (OKD)
- Gradle and Node Package Manager (NPM) build system
- Jenkins, Nexus, SonarQube, and Docker CI/CD pipeline

BUSINESS VALUE

Coveros was able to help PECOS achieve a number of business benefits, including increased ability to implement system improvements while reducing risk, escaped defects, and test cycle time.

With help from Coveros, PECOS was able to develop over a hundred functional and API tests and get them running within their continuous integration pipeline. This allowed rapid feedback about application health and feature quality in an automated fashion, all before code was ever merged. Thanks to the design of the tests and framework, these tests could be executed quickly, allowing dozens of tests to run in under a minute. This rapid feedback gave developers more confidence to move forward with their development.

Thanks to the containerization of the test cases, testing was no longer isolated to individual QAs, and the quality of the application could be verified early on in the development process—even allowing devs to test before pushing their code. This additional confidence in the software integrity sped up the development process and pushed testing left.

CONNECT WITH COVEROS

coveros.com

info@coveros.com

929.341.0139

twitter.com/coveros

linkedin.com/company/coveros