Quality Assurance Improvements

Our client opted to implement a web application to facilitate the support and monitoring of operations involved in the extraction of various materials in mining processes. The development team initially consisted of nine seasoned software developers, three interns, and one QA engineer. As the project progressed, the application began to deliver substantial increments, which became too large to be managed effectively by a single tester. In response, our company assigned two additional experienced QA engineers to ensure the testing process remained seamless and maintained a high standard of quality.

Utilized Technology Stack

Cloud: Microsoft Azure

Backend Framework: MongoDB Atlas, GraphQL, .NET

Frontend Framework: AngularJS

Test automation framework: Cypress.io, Cucumber

Review of Challenges

When we joined the project, it had already been underway for several months. The infrastructure had been set up, the basic application framework implemented, and the testing processes established within the company. Automation tools, including a BDD (Behavior-Driven Development) layer, had already been configured, and a few automation tests had been created.

At that point, the priority was to test and document the tasks in the backlog, subsequently automating them. Regression tests were scheduled to run once per sprint. Automation test scripts were stored in separate repositories, each assigned to an individual QA engineer. While a basic pipeline had been configured, test runs were triggered manually rather than automatically. Additionally, the existing automation tests were not fully isolated, with some tests depending on the results of previous tests.

The automation test suites were set up to run exclusively on the pre-production environment, which was designated for regression testing and deployed with stable code once per sprint.

Our Solution

Process changes

During the first regression testing cycle, the QA team identified a significant issue: the regression bugs that were reported were not easily visible to the wider team. There was no mechanism to organize or prioritize these bugs, as the existing process only allowed bugs to be linked to User Stories within the current sprint. To address this, we updated the process by introducing a separate User Story specifically designated as a “bucket” for regression bugs. This change improved visibility for the entire team and streamlined the bug triage process, making it more transparent and easier to manage.

At a later stage in the project, the development team (including both developers and QA professionals) repeatedly raised concerns about the readiness of User Stories for development. Issues included missing or unclear design specifications and incomplete or ambiguous acceptance criteria, which sometimes did not cover all the requirements outlined in the User Story. In response, our QA experts recommended establishing clear “Definition of Ready” (DoR) and “Definition of Done” (DoD) criteria for each User Story. This initiative significantly improved the quality of the User Stories, reduced development time, and led to fewer redundant bugs, as the team had a clearer understanding of the scope and requirements from the outset.

Additionally, running automation tests exclusively in the pre-production environment proved inefficient. Regression bugs were often identified late in the process, and fixes for some of these bugs had to be deferred to the next sprint. This delay could have been avoided if the bugs were reported earlier. To address this, we updated the automation test setup to run on the test environment and configured the tests to trigger automatically every night. This change allowed us to receive daily feedback, enabling the team to catch regression bugs earlier in the development cycle, thereby improving the stability and quality of the application. The pipeline was also reconfigured to run tests in the pre-production environment when the regression testing was performed, ensuring comprehensive coverage across both environments.

Security

Users and automation tools must use logins and passwords to access the application. Storing them as plain text in the repository, even a private one, is a security vulnerability. With our expertise, we reconfigured the CI/CD setup to use encrypted and hidden from the view passwords to provide maximum security to the solutions. All sensitive data has been moved away from access by unauthorized users.

Test efficiency

As the automation test framework evolved and new tests were continuously added, we identified several opportunities to improve its efficiency. We began by consolidating all test repositories into a single repository, which enabled us to create and share common methods and tests, ensuring a unified source of truth.

In collaboration with the Team Lead, we introduced several key improvements to further enhance the framework:

1. **Elimination of Test Dependencies:** We decoupled the tests so that each test could run independently. This increased the reliability of the automation tests, allowed tests to be executed in isolation, and reduced the time needed to debug test failures.
2. **Session Management**: We introduced a mechanism to store sessions between tests. This significantly reduced test execution time by eliminating the need to log in at the start of each test.
2. **Refactor of Waits:** We replaced general wait times with targeted waits, where the tests pause to ensure specific elements are loaded or server responses are received. This change improved the speed of the test runs and eliminated inconsistencies, resulting in more stable and reliable test outcomes.
3. **Enhanced Pipeline Configuration:** We updated the pipeline configuration to allow the triggering user to specify parameters such as the environment and browser to be used for the test run. This added flexibility and control over the test execution process.
4. **Support for Running Individual Tests:** The pipeline configuration was modified to support running individual tests. This allowed QA engineers to rerun failing tests more easily, making the debugging process more efficient.
5. **Publishing Test Results and Screenshots:** We added the capability to publish test results and screenshots, providing QAs with immediate feedback on test outcomes. This feature enabled faster identification of failure reasons and streamlined the debugging process.
6. **Flakiness Fixes:** We identified flaky tests, resolved the underlying issues causing the instability, and implemented safeguards to prevent future occurrences of flakiness. This has helped improve the overall reliability of our test suite.

These improvements collectively enhanced the automation test framework's efficiency, reliability, and maintainability, allowing for faster development cycles and higher-quality outputs.

Subsequent Outcomes

Quality Assurance (QA) is not just about testing; it's about setting up the right foundation for product development from the very start. An experienced and professional QA engineer plays a critical role in establishing the quality processes early in a project, leading to several key benefits:

- **High-quality product from the outset:** With QA involved from day one, the product quality is ensured throughout development.
- **Faster and more efficient automation test development:** Automation test portfolios can be built more quickly and easily.
- **Stable and reliable regression suites:** Regression testing becomes more effective and dependable.
- **More time for manual testing and tool implementation:** The team can allocate more time to manual testing and integrate additional tools (such as API testing) to further enhance quality.

Results from over a year of experience on this project demonstrate the impact of a Consultant QA engineer. Compared to a regular tester on the client side, the Consultant QA engineer:

- reported **15% more bugs,**
- tested **40% more User Stories,**
- created 440% more automation test scripts, **leading to over 90% automation test coverage.**

These outcomes were only possible thanks to a team that embraced continuous improvement and trusted the QA engineers' recommendations for process changes.

"Quality assurance isn't just testing — it's the foundation for delivering reliable software at scale."

WE'RE BUILDING BETTER DIGITAL PRODUCTS