Mykyta Knysh — Portfolio

Email: nknysh@gmail.com | LinkedIn: www.linkedin.com/in/mykytaknysh
Resume: https://nknysh.github.io/cv/ | Demo: https://github.com/nknysh/cmc-tests

PORTFOLIO

QA Lead — Introduced the first QA function and moved release verification to CI/CD with around 2,000 automated tests

CHALLENGE

The client's company had no formal quality assurance and no QA role. Requirements and defects were not tracked in any consistent way, and nobody could say with confidence whether the software was ready before a release. Every release carried unknown risk, with no evidence to support or challenge the go decision.

MY SOLUTION

- Introduced the QA function to the organization and built and led the team of manual and automated testers (up to 10 people), coordinating QA efforts across it. I hired and onboarded the testers and mentored them.
- Put in place tracking for requirements and defects in Jira, with Zephyr for test management, so that what was expected, what was found, and what was fixed became visible to the team.
- Defined the process around it: a defect life cycle with triage, the regression scope, and test metrics, so defects, regression checks, and quality reporting followed one agreed process.
- Tested hands-on before releases across the web and iOS user interfaces, APIs, databases, patches, logs, third-party service integrations, and performance.
- Added test automation for UI and API checks and integrated it into CI/CD with TeamCity, so release confidence no longer depended on manual effort alone.

RESULT

Release verification moved from ad hoc checks to a CI/CD pipeline running around 2,000 automated tests within 1 hour, backed by tracked requirements and defects, with a QA team of up to 10 people in place.

TECH STACK

Jira, Zephyr, TeamCity, Nightwatch.js, Cypress, JavaScript, JMeter, Postman

Lead Test Automation Engineer — Built the test strategy, test automation framework and CI pipeline from scratch for a new product and team

CHALLENGE

The product and the team were both new, and I joined as the lead test automation engineer with no testing strategy or approach to build on. It was unclear what to test, at which level, and how much to automate, so there was no basis for deciding where testing effort should go.

MY SOLUTION

- Communicated closely with the product owner and the development lead to clarify requirements and the product architecture, and used that understanding as the basis for the test strategy.
- Defined the test strategy and a test automation roadmap based on product risks, business priorities, technical feasibility, and ROI, so automation effort went first to where it paid off most.
- Set up test automation and CI from scratch and built the suites across layers: both UI and API tests with Playwright, with the API tests covering WebSocket interfaces.
- Automated the highest-risk flows first: critical user scenarios for the UI, plus basic and security-related tests for the API, so the most important behavior was covered early and coverage was extended from there.
- Grew the team to 6 people: interviewed test automation engineer candidates, then mentored the engineers and built a culture of learning and improvement in the team.

RESULT

The team went from no testing approach to a risk-based roadmap and automated UI and API tests for the microservice-based product running in CI with test reporting, growing into a combined suite of more than 3,600 UI and API tests.

TECH STACK

TypeScript, Playwright, WebSocket, Docker, GitLab CI, Postman, Jira, ESLint, SonarQube

---

Senior Test Automation Engineer — Eliminated test flakiness and cut UI test execution time by 50% in a Playwright suite of 1,100 tests

CHALLENGE

The Playwright test suite had flaky tests and no established monitoring of test results. When a test can fail without a real defect, a red run may mean a product problem or just noise, so failures cost time to investigate and the results are harder to trust. Long UI test runs also delayed feedback on every change.

MY SOLUTION

- Established test result monitoring with Allure reporting, so results were visible and unstable tests could be identified and fixed.
- Replaced setup hooks with fixtures to reduce flakiness, so each test runs only the setup it needs and nothing more.
- Put web-first assertions to correct use, so tests wait for the expected state instead of checking it at a single moment.
- Ordered triggers and listeners correctly, registering each listener before the action that triggers the event, so tests do not miss a fast response.
- Mocked third-party services, so UI tests no longer depend on the speed or availability of external systems. Moved testing of the real integrations to a separate suite.

RESULT

Across a suite of 1,100 UI and API tests, no flaky tests remained and test execution time for UI tests was reduced by 50%, which supported further growth of a healthy test suite.

TECH STACK

Playwright, TypeScript, Allure

Test Team Lead — Unified QA practices across four Scrum teams and cut escaped defects by 70-85%

CHALLENGE

Testers in four Scrum teams of a large software product each worked in their own way, with no shared test process, techniques, or quality reporting, so quality could not be compared or steered across teams. Testing relied on scripted cases that confirmed documented behavior, so edge-case bugs and defects in lightly documented areas escaped to production.

MY SOLUTION

- Led a group of 10 testers embedded across the four teams and owned one shared test process, checking that it was followed or improved, instead of four separate approaches.
- Introduced test design techniques that do not depend on complete documentation: ACC modeling and pairwise testing to cover feature and parameter combinations systematically, and structured exploratory testing with deliberate attempts to break the software in under-documented areas.
- Used statistics and metrics to spot problems across teams and track their resolution, so changes were driven by evidence, not opinion.
- Ran root cause analysis (RCA) on defects that escaped to production, found where process or test design let them through, and fed each finding back into the shared process and the test design techniques.
- Stayed hands-on, taking part in testing and managing testing of MS Office add-ins and web applications, and mentored and coached the testers on the new techniques so the practices spread through people, not just a document.

RESULT

The four teams now shared one process, one set of techniques, and comparable quality data, and the number of escaped defects per release dropped by 70-85%.

TECH STACK

MS TFS, MS Test and Lab Manager, MS Hyper-V, MS DWH, Excel pivot tables, MS SharePoint, MS Office

---

QA Lead — Grew test automation from 50 to 2,200 tests on an up-to-date framework owned by the test team

CHALLENGE

The product's test automation framework was outdated and covered very little of the application. The automated tests had been written by software developers without a systematic approach, so the suite was hard to extend and coverage stayed low. Regression confidence depended on a small set of checks that nobody on the testing side owned.

MY SOLUTION

- Built a proof of concept to show that a modern framework would work for this product before committing the team to it, then introduced Cypress as the updated test automation framework.
- Set the structure around industry best practices: the page object pattern, fixtures, and reusable Cypress procedures, so new tests reuse existing building blocks instead of starting from scratch.
- Moved ownership of test automation to the testing team, so coverage was planned and grown systematically instead of ad hoc.

RESULT

Grew the number of automated tests from 50 to 2,200, with coverage now planned and owned by the testing team.

TECH STACK

Cypress, JavaScript

Lead Test Automation Engineer — Covered a new reusable library of interactive UI elements with 900+ visual tests and functional UI tests

CHALLENGE

A new reusable library of visual, interactive elements was being built to serve as a shared UI foundation. A defect in one shared element would appear in every product that used it, and it could be either a broken interaction or a visual change that no functional check would notice. The library changed continuously while it was being built, so manual checking of how every element looked and behaved could not keep up.

MY SOLUTION

- Split the testing into two complementary layers and defined what each one had to prove: functional UI tests for how elements behave, and visual testing for how they look.
- Automated functional UI tests with Playwright to check element behavior and interaction, so a change to one element did not silently break how it worked.
- Added visual testing, also with Playwright, based on a snapshot approach: each element's rendering is compared against an approved snapshot to catch unintended changes in how elements render, which functional assertions do not detect. The trade-off is that every intended design change also fails the comparison, so its snapshots must be reviewed and updated.
- Integrated both suites into CI so every change to the library was checked automatically, with test reporting.

RESULT

Created more than 900 visual tests, run in CI alongside the functional tests, so unintended behavior and appearance changes were caught before products consumed the library.

TECH STACK

TypeScript, Playwright, React, GitLab CI, Jira, ESLint