Quality Assurance & Technical Staffing Since 2024

Engineering Confidence:
Our Approach to Software Quality Assurance

A pacesetter sets the tempo at the front of the field. We do the same for quality: catching risk before it slows a release down, not after.

Executive Summary

Pace Setter Solutions LLC provides specialized software Quality Assurance (QA) and technical staffing for organizations that cannot afford to ship broken software, from federal agencies and prime contractors to early-stage technology startups. We specialize in the validation of complex requirements by pairing deep testing expertise with staff who communicate clearly and get up to speed fast. That combination has kept us working with the same federal program offices and prime contractors across multiple contract cycles, not just single engagements.

This paper covers our philosophy and the methodology behind how we work, so you know what to expect before a project starts.

Our Philosophy: “Quality-as-a-Service (QaaS)”

Instead of treating testing as a final checkpoint before release, quality checks and gates should be built into every stage of development through a Shift-Left philosophy. Risks and defects should be caught as early as possible, before they get expensive to fix.

In practice, automated quality gates are built directly into an application’s Continuous Integration/Continuous Delivery (CI/CD) pipeline to validate application integrity as code is deployed up the stack of environments in the Software Development Life Cycle (SDLC). Automated quality gates should also include Performance Testing or Load Testing before peak traffic ever reaches production environments.

Pace Setter Solutions’s greatest investment is always in our people. We ensure every consultant is trainedwhat is on current tools and practices, because we've built our reputation on professionals who pair technical depth with strong communication skills. Quality, trust, and accountability aren't slogans here; they're what the methodology below is built to enforce.

The Full Stack QA Model

Most QA organizations split testing across separate specialists for manual testing, automation, performance, and security. We build our teams differently. Every engineer we place is a Full Stack QA: one person who can write a unit test in the morning, automate an API suite in the afternoon, and run a load test before end of day.

The easiest way to picture the role is next to one most people already know: the Full Stack Developer. A Full Stack QA engineer covers that same stack, just from the testing side:

Skill Area	Full Stack Developer	Full Stack QA Engineer
Front End	Builds the UI components and user flows	Automates and validates those same UI flows end to end
APIs & Services	Writes the services and business logic	Builds automated API and integration checks against that logic
Data Layer	Designs and models the database	Verifies data integrity, migrations, and edge cases in that database

Skill Area	Full Stack Developer	Full Stack QA Engineer
Pipeline	Deploys code through the CI/CD pipeline	Builds the automated quality gates that run inside that pipeline
Under Load	Optimizes code for performance	Load- and stress-tests the system to confirm it holds up

Same stack, different job: one side builds it, the other proves it holds up.

That range is exactly why we don't build teams out of manual testers alone. As an application grows, the number of test cases needed to cover it grows with it, and QA demand climbs along the same curve. A team stuck repeating the same manual steps release after release can't keep pace, and QA turns into the bottleneck it's supposed to prevent. Every engineer we place builds toward automation as part of the job, not as a skill to backfill later.

That model means a smaller team covers more ground, with fewer handoffs and less time lost explaining context between specialists. It's also why we invest so heavily in continuous training. A Full Stack QA engineer only works if that engineer stays current across every layer of the stack.

The Four Pillars of Testing: Integration, Verification, Performance and Validation

We rely on four core pillars. Each one exists to reduce risk before code is deployed to production:

A. Integration

We design and deploy UI and API automation frameworks built for high-availability systems, scaling automation coverage from zero to 85%+ on legacy-to-cloud transitions and reducing manual regression cycles from hours to minutes.

B. Data Integrity & Migration Verification

Large-scale data migrations to the cloud often require custom automated validation engines that replace manual spot-checking. These engines run bit-for-bit reconciliation across hundreds of thousands of records in parallel, so no data gets lost during high-consequence transitions.

C. Performance Engineering & Scalability

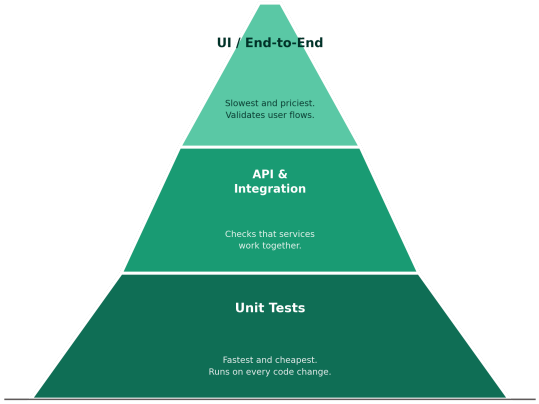
Load and stress testing are essential pieces of the lifecycle to find bottlenecks before they reach production. We simulate peak traffic and enrollment periods, then verify the system can absorb multiples of standard load without downtime. In addition, we use stress testing to simulate what happens to an application when things begin to fail, so that DevOps teams can understand where their single points of failure reside and eliminate them to improve system reliability and eliminate costly downtime.

D. Mission-Critical Compliance and Validation

We make sure every deployment meets regulatory and accessibility mandates without slowing the development cycle, including full Section 508 accessibility sign-off, HIPAA, NIST SP 800-171, Zero Trust, and many others as the environment demands.

The Testing Pyramid

Not all tests are equal. Where applicable, a minimum of three layers of automated checks should be included. Automated checks should be prioritized by running the fastest and cheapest tests most often and the slowest and most expensive tests only where they add real value.



More tests at the base, running in seconds. Fewer tests at the top, running in hours.

Unit tests check a single function or component in isolation, and they run in seconds. A layer up, API and integration tests confirm that separate services talk to each other correctly. At the top, UI and end-to-end tests drive the whole application the way a real user would, clicking through complete workflows, which is why there are fewer of them. The base should be very wide and broad and the top narrow: a thousand slow browser tests would only catch the same bugs a hundred fast unit tests could have caught first.

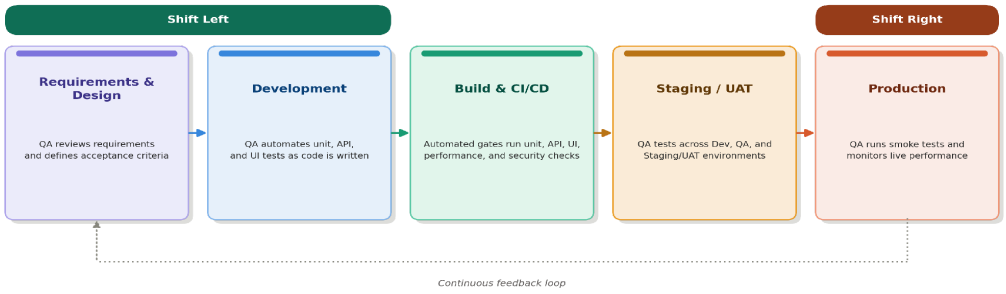
Types of Functional Testing

Beyond the pyramid layers, test plans cover several categories of functional testing, each answering a different question:

- Smoke testing: does the build even work well enough to test further?
- Regression testing: did a new change break something that used to work?
- System testing: does the fully integrated application meet the requirements end to end?
- User acceptance testing (UAT): does the software solve the business problem, from the user's point of view?

Where QA Fits In

As demonstrated earlier, quality checks into every stage of the development lifecycle, from initial requirements through production monitoring, so nothing gets left for the end:



Independent Verification & Validation (IV&V)

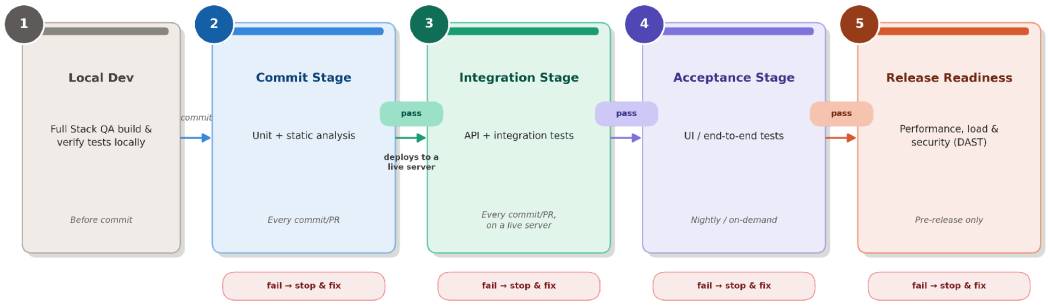
Independent Verification & Validation keeps testing accountable to someone other than the person who wrote the code, and it maps directly onto the lifecycle above. Verification, confirming we built the product right, happens during Development and Build & CI/CD: static analysis, unit tests, and API/integration checks, run by QA engineers who did not write the code under test. Validation, confirming we built the right product, happens in Staging/UAT: system and user acceptance testing that checks the finished application against the original business requirement. That independence carries into Production too, where post-deploy smoke checks and monitoring close the loop back to what was asked for in Requirements & Design.

Requirements Traceability

Every requirement should trace forward to the tests that validate it, and every test result should trace back to the requirement it proves. We implement that discipline in a test case management tool by tagging each automation and regression test with the work item or requirement number it covers. That tag travels with the test through every run, so when a requirement changes, it's immediately clear which automated tests need a second look before the next release, and a program manager or auditor can walk from a single requirement straight through to the test evidence that satisfies it.

CI/CD Quality Gates

Before any of this starts, Full Stack QA engineers build and verify tests locally, catching bad developer logic before code reaches a shared server. From there, code earns its way through four gates in order: unit and static analysis, API and integration, UI, then performance and security. The first gate triggers a deployment to a real, live server, so gate two onward runs against that live deployment, not a local sandbox. A failure at any gate sends the build back for a fix instead of letting it slide through.



That order sets the cadence too: unit and API tests run on every commit since they're fast, UI tests run nightly or on demand instead of blocking every check-in, and performance and security only make sense right before a release.

That doesn't say which environments run which tests, though. Every environment from Dev through Staging/UAT runs the same core suite of unit, API, and UI tests. Staging/UAT also adds performance and security testing, since it sits closest to production, often in the same Virtual Private Cloud (VPC). Production gets a lighter, post-deploy version of that suite: smoke-level unit, API, and UI checks confirming the release works once it's live.

It is important to note here that performance and security are the last set of tests to run in each environment as code moves up the stack. These tests are expensive, complex, and require a lot of time to analyze results and come up with action plans for mitigation. If Performance or Security tests are performed before the application has been validated by the other tests, the Performance or Security tests could have been performed on an invalid application version, forcing those expensive and time consuming Performance and Security tests to be run again. By having them at the end, we ensure we aren’t wasting testing cycles for Security or Performance on an invalid version of the application.

Test Type	Dev	QA	Staging / UAT	Production Post-deploy smoke
Unit + Static Analysis Every commit	✓	✓	✓	✓
API + Integration Every commit	✓	✓	✓	✓
UI Tests Nightly / on-demand	✓	✓	✓	✓
Performance Testing Pre-release only	–	–	✓	–
Security Testing (DAST) Pre-release only	✓	✓	✓	–

Right-Sizing Your QA Team

Understaffing quality is one of the most common ways organizations set themselves up to fail. Based on our engagements, having the right amount of QA in place, sized to the pace and complexity of the release schedule, matters more than headcount alone. There is no single formula that fits every organization, so we work with each client to size the team correctly rather than applying a one-size-fits-all number, keeping QA from becoming the bottleneck as applications expand and scope increases.

Proof in Practice

Two engagements show this method in practice:

Federal Healthcare Agency: Cloud Data Migration

A federal healthcare agency needed to move millions of healthcare records to the cloud, rolling the migration out state by state. Fourteen Scrum teams worked the broader modernization effort, but the migration itself belonged to one team, who owned verifying that every record moved over completely and correctly at each stage of the rollout.

We staffed that team with Full Stack QA engineers who could validate the data, drive the UI, and load-test the pipeline all at once, so coverage could scale with the rollout instead of waiting on new specialists at every stage. The first state went live with 171 test cases, run entirely by hand, on a four-day execution cycle. By six states, the same coverage ran as 187 tests, 90% automated, in five minutes. At thirty-nine states, it had grown to 197 tests, 95% automated, running in forty-five minutes inside a CI/CD pipeline, though a one-to-two-day manual pass was still needed at that stage to check the handful of data points that couldn't be automated for one reason or another. By the time all fifty-nine states and territories were live, the suite covered 200 test cases, fully automated and running in parallel, closing that last manual testing gap and running gate to wire inside that same forty-five-minute CI/CD run.

That automation reconciled records field by field at the data layer, while the same engineers' API tests confirmed the data was reachable and correct at the service layer well before anyone needed a full UI pass to catch it, and their UI and performance tests confirmed the application held up as more states came online. Coverage that once took four days for a single state, run entirely by hand, now runs coast to coast in forty-five minutes with no manual step left, without adding headcount as scope grew.

Large-Scale Enterprise Infrastructure: Regression Automation

A high-availability enterprise system had no automation coverage at all, so every regression cycle slowed the team down. We staffed this engagement with Full Stack QA engineers rather than siloed specialists, applying Shift Left and Shift Right in the same pipeline: catching regressions early in development, then validating performance and stability again once code reached production. The same engineers who built the regression automation suite also owned API testing and performance testing, using the same framework and methodology across all three. The result: 500+ daily regression tests running with a 99% success rate, cutting developer downtime by 15% per cycle, with performance regressions now caught before release ships.

Working With Us

Whether you're a federal program office vetting a prime contractor's QA partner or a startup building your first automated test suite, our approach doesn't change. We learn your mission, build quality in from day one, and keep it visible to every stakeholder, from program managers to engineers. Many of our engagements start as a single project and grow into a standing partnership, which is the outcome we aim for from day one.

If that approach fits what you're looking for, we'd welcome the conversation.

Request a Technical Consultation →