



Overcoming the Test Automation Gap

How Mature QA Teams Reduce Manual Testing, Flaky
Tests & Automation Bottlenecks

Table of Contents:

Introduction: The Test Automation Gap Is a Chasm	3
The State of Test Automation Today	5
Current Automation Rates vs. Expected Automation Rates	
What's Blocking Progress?	
Why the Automation Gap Matters Now	
Automated Testing Types	
Average Number of Automated Tests Run Daily	
Continuous Integration/Continuous Delivery (CI/CD) Adoption	
What Teams Are Trying to Fix Next	
Why QA Teams Struggle to Automate More Testing	15
Skill & Expertise Gaps	
Applications Under Test (AUT) Complexity	
Tool Fragmentation & Lack of Centralized Management	
Time & Resource Constraints	
Test Stability & Flakiness Issues	
Inadequate Test Coverage Strategies	
Lack of Executive Buy-In & Organizational Support	
AI Is Increasing the Cost of Staying Stuck	
How Mature QA Teams Close the Gap	19
Assess Your Current Automation Maturity	
Build Skills & Expertise	
Choose the Right Tool for the Job	
Build Reliable, Resilient Automated Tests	
Strengthen Cross-Team Collaboration	
Connect Automation & Test Management	
Use AI to Strengthen Test Automation	
Measure, Scale & Sustain Automation	
The Future of Test Automation in the Age of AI	29
AI-Driven Test Automation	
Shift-Left Testing	
Better Integration Between DevOps & Test Automation	
Closing the Test Automation Gap	31
Take the Next Step	

The Test Automation Gap Is a Chasm

[Test automation](#) is hardly a new technology; many of the frameworks and tools we use today first emerged on the market over 20 years ago. Yet nearly everyone who uses automated testing still has trouble with the test automation gap, and for most teams, that gap does not appear to be closing.

Simply put, the test automation gap separates what teams want to automate from what they can realistically automate today.



The test automation gap, defined

The test automation gap is the difference between the testing a team wants or needs to automate and the testing it can realistically automate with its current tools, processes, skills, and capacity.

Before AI, most teams already prioritized test automation, yet many didn't make progress. Easy tests, like repeatable regression tests and straightforward functional checks, were automated first. More complex tests remained manual, became too expensive to maintain, or grew flaky and unreliable as their complexity exceeded the capabilities of the automation framework, tooling, or the engineers building them. The persistent gap between automation goals and reality remained, but for many teams it still felt sustainable.

Then AI-generated code changed everything. As development velocity increases, so does the volume of code that needs to be tested. AI-generated code can also introduce additional quality risks, particularly when teams rely on it without the same level of review and testing applied to human-written code. At the same time, QA engineers are increasingly expected to use AI-assisted tools without established practices for reviewing their output, setting appropriate guardrails, or managing inconsistent results. The automation gap that teams have lived with for years is widening into a chasm between what they can test and what they need to test.

[Our annual industry research](#) has consistently shown QA teams falling short of their automation goals. Between 2020 and 2025, despite setting ambitious targets, actual automation rates were around 40%. More recent data now shows that **57% of QA tests are automated, based on respondents' estimates**. This is progress, but it also means that a substantial share of testing remains unautomated, even as AI increases development speed and testing demand.

QA teams have more advanced tools and efficient methodologies than ever before. So why does the test automation gap persist, costing time and test coverage while limiting teams' ability to scale and keep up? That's the question we're here to answer.

Although it may be a comfort to know you're not alone, just because the test automation gap is a widespread problem doesn't mean it's not a critical one. **Consistently falling behind on automation goals significantly impacts software quality, release speed, and team efficiency.**

This ebook—designed for QA engineers, test automation engineers, QA managers, software development leaders, and software developers—provides actionable insights and strategies to accelerate automation so you can reduce the gap, improve coverage, and build a sustainable automation program.

Section 1

The State of Test Automation Today

The State of Test Automation Today

Test automation has advanced, but progress remains uneven. Teams are automating more of their testing while still dealing with complex workflows, limited resources, unstable tests, and disconnected development processes.

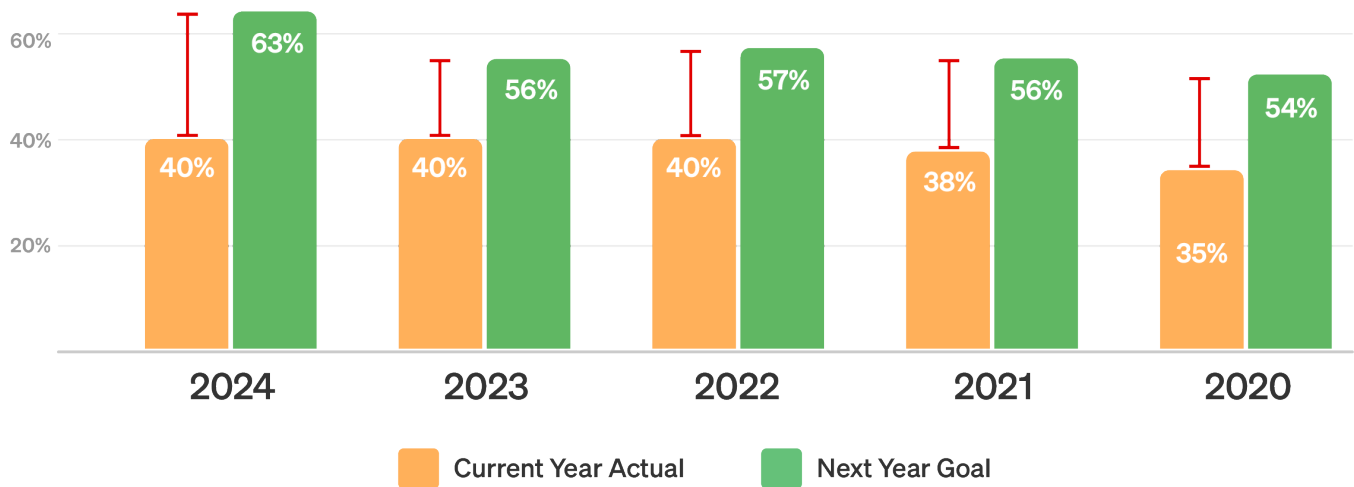
Sembi's [2026 Software Quality Pulse Report](#) contains the latest view of this landscape. Based on responses from nearly 4,000 QA engineers, security professionals, developers, and engineering leaders, the report examines how teams approach software quality and automation today, what prevents them from making greater progress, and where they plan to invest next.

Its predecessor, TestRail's [Software Testing and Quality Report](#), provides historical context. This annual benchmarking series surveyed thousands of QA and DevOps professionals over the past half-decade about their tools, practices, and priorities. While its benchmarks do not yet have a direct Pulse Report equivalent, figures from the 2025 edition have been retained here and identified as historical data.

Together, these reports provide a current snapshot and a longer-term benchmark for understanding where test automation stands and how your QA team compares with its industry peers.

Current Automation Rates vs. Expected Automation Rates

What percentage of tests do you expect to be automated vs. manual in the next year?



Nothing illustrates the historical test automation gap better than this year-over-year comparison of expected and actual automation rates.

Teams' goals rose steadily, with a 7% jump in the 2025 target. **Actual automation rates increased much more slowly and remained around 40%, leaving teams unable to keep pace with their growing automation ambitions.**



57% of QA tests are automated in 2026

The Software Quality Pulse Report shows that teams have made progress. However, 43% of QA testing remains unautomated. With automation already well underway, the next stage of maturity means tackling the complex, brittle, business-critical workflows that remain manual.

What's Blocking Progress?

The Software Testing and Quality Report also asked QA teams about the pain points and challenges with automation:

- ◆ **Test reliability and maintenance:** Frequent test breakages (45%), flaky or unreliable tests (26%), slow test execution (21%)
 - ◇ Inconsistent test results slow teams down and make it harder to trust automation as part of the CI/CD pipeline.

- ◆ **Automation capacity and coverage:** Test data management (32%), lack of skilled personnel (30%), inadequate test coverage (26%)
 - ◇ Many teams say they still cannot automate enough of their testing, leaving gaps in their quality assurance strategy.
- ◆ **Tools and workflow integration:** Tool selection challenges (29%), limited CI/CD integration (19%)
 - ◇ QA engineers don't always have a say in tool selection, leaving teams with frameworks that may not fit their applications, skills, DevOps workflows, or testing needs.

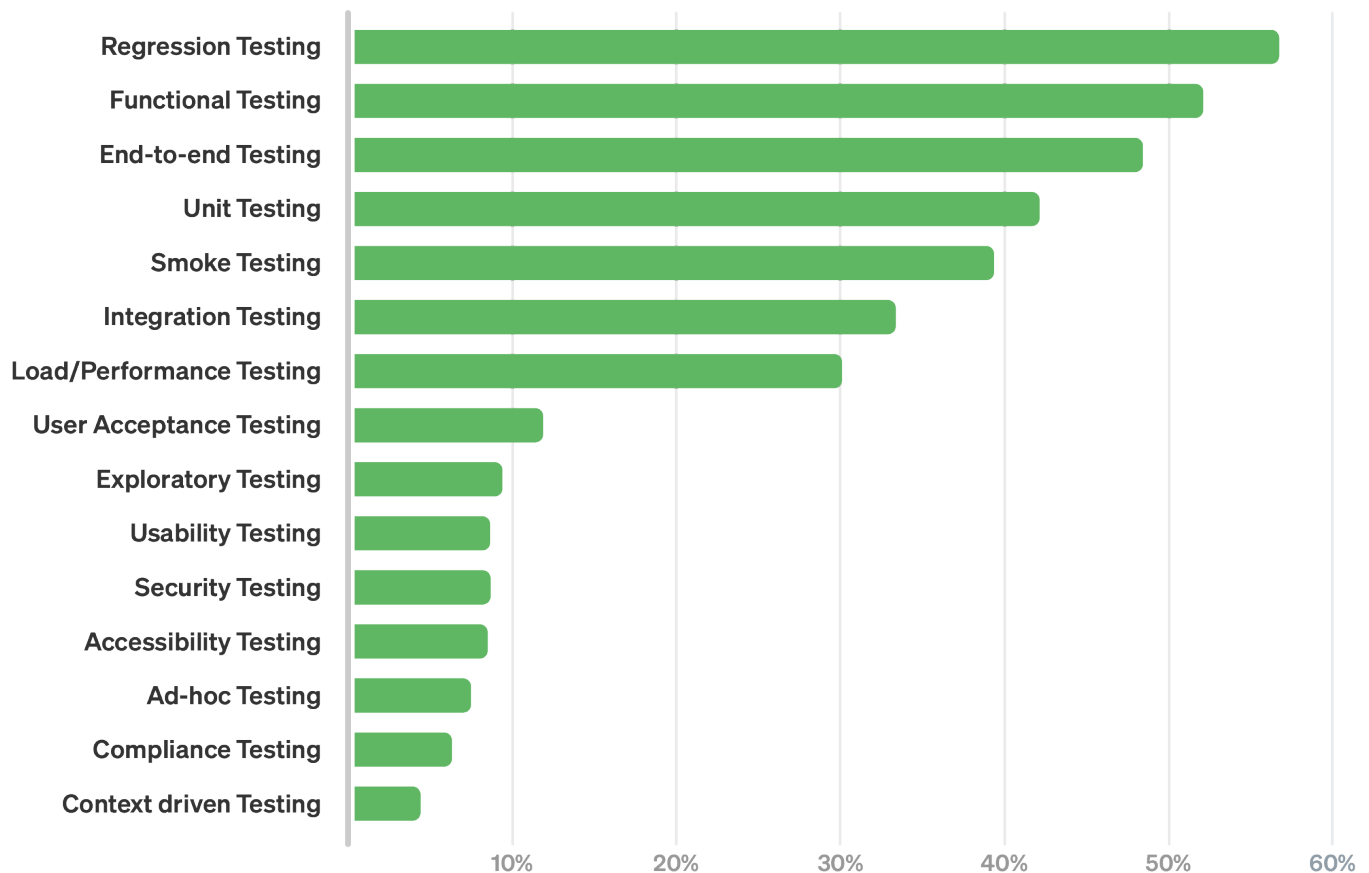
These challenges show why adopting automation is not enough. **Teams must also maintain their tests, integrate automation into development workflows, build skills, and expand coverage without introducing delays or instability.**

Why the Automation Gap Matters Now

AI capabilities are the leading 2026 investment priority for 35.7% of teams, compared with 20% that identify QA automation as their top priority. That imbalance raises the stakes. As AI accelerates code production and change, the tests used to validate that code must keep pace. Without corresponding progress in QA automation, AI investment could widen the gap between development output and testing capacity.

Automated Testing Types

What kind of testing does your team run with automation?



Survey responses show that most teams focus on automating tests that deliver high impact and are easy to repeat, such as regression and functional testing. These tests can be time-consuming to conduct manually, so automating them speeds up release cycles and frees testers to focus on work that requires human insight.

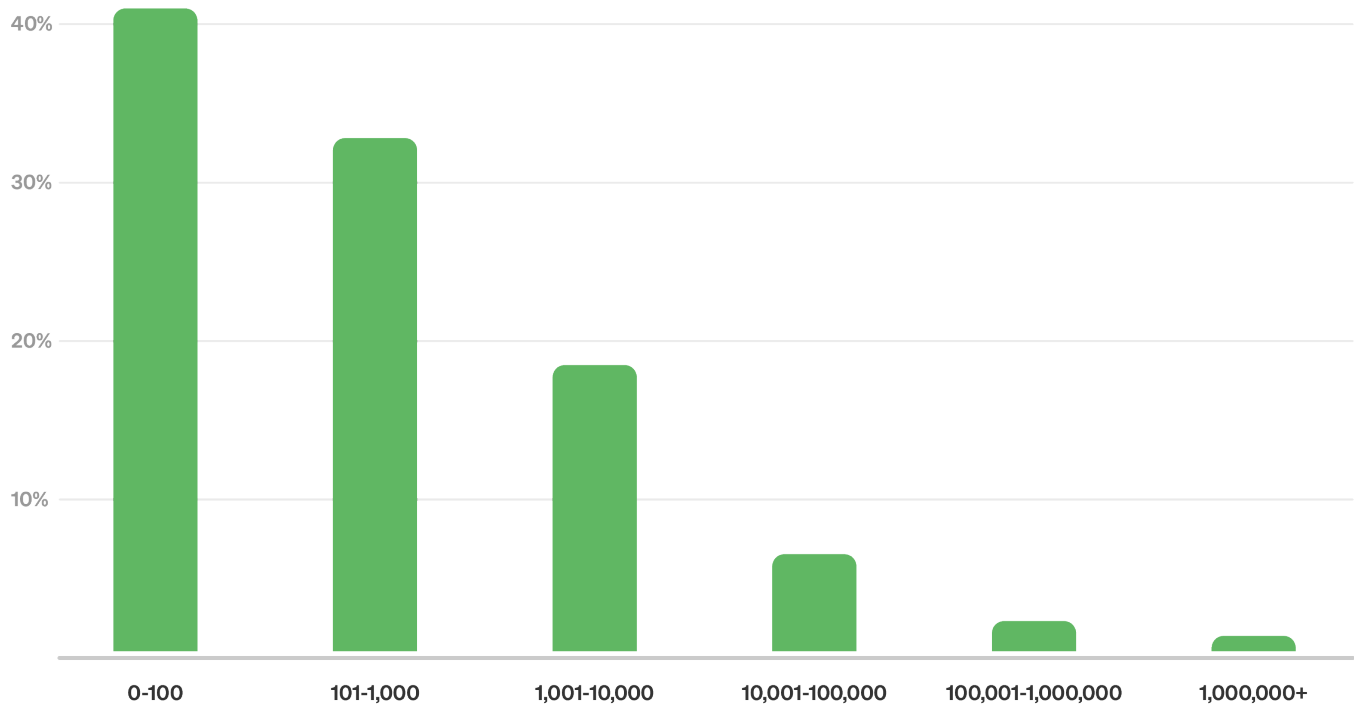
It might sound surprising, but 100% automation is not and should never be the goal. Testing disciplines such as exploratory, usability, accessibility, security, and compliance testing often require human intuition, perception, or creativity that automation cannot replicate. Automated checks can support these efforts, but trying to automate every single aspect may create more maintenance work than it eliminates.

An efficient test automation program focuses on automating the right tests, not every test.

The automation gap isn't created because teams fail to automate easy tests. In many cases, those are the first tests to be automated. The gap grows when teams move beyond repeatable regression and functional testing into more complex workflows, dynamic UIs, cross-platform scenarios, and business-critical paths that are harder to automate and maintain.

Average Number of Daily Automated Tests

On average, how many tests does your organization run each day?



This chart illustrates the wide range of test automation scale.

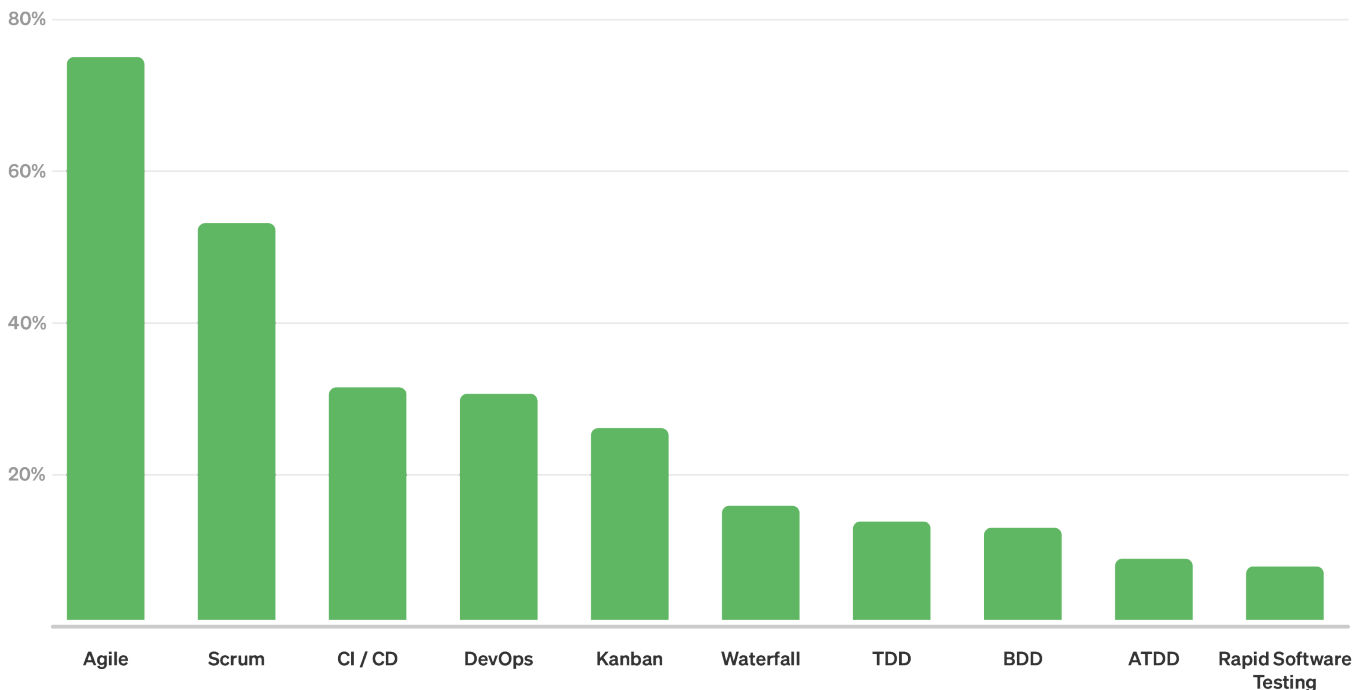
Most teams remain toward the lower-to-middle portion of that range. The Software Testing and Quality Report found that about **41% execute fewer than 100 automated tests per day, while another 33% execute between 101 and 1,000**. These figures reflect daily test execution volume rather than the total number of tests a team has automated.

The right-hand side of the chart demonstrates the test volume that a sufficiently scaled automation infrastructure can support. However, **volume alone does not indicate automation maturity**.

Daily automated test volume is just one signal of maturity. Mature QA teams also measure test reliability, coverage of high-risk workflows, maintenance, CI/CD integration, and their confidence in the results of automation to release faster.

Continuous Integration/Continuous Delivery (CI/CD) Adoption

Does your team use any of the following development methodologies or techniques?



[CI/CD is a critical piece](#) of the test automation puzzle. Integrating automated tests into the CI/CD pipeline allows teams to run tests with every code change, enabling faster, higher-quality deployments while saving time for testers and developers.

The 2026 Software Quality Pulse Report provides a more current view of this integration gap. Only **26% of teams report that their QA tools are mostly or fully integrated within DevOps pipelines, leading to limited visibility, disrupted feedback loops, and slowed decision-making.** The remaining teams report partial integration or siloed processes, limiting their ability to receive consistent, automated feedback throughout development.

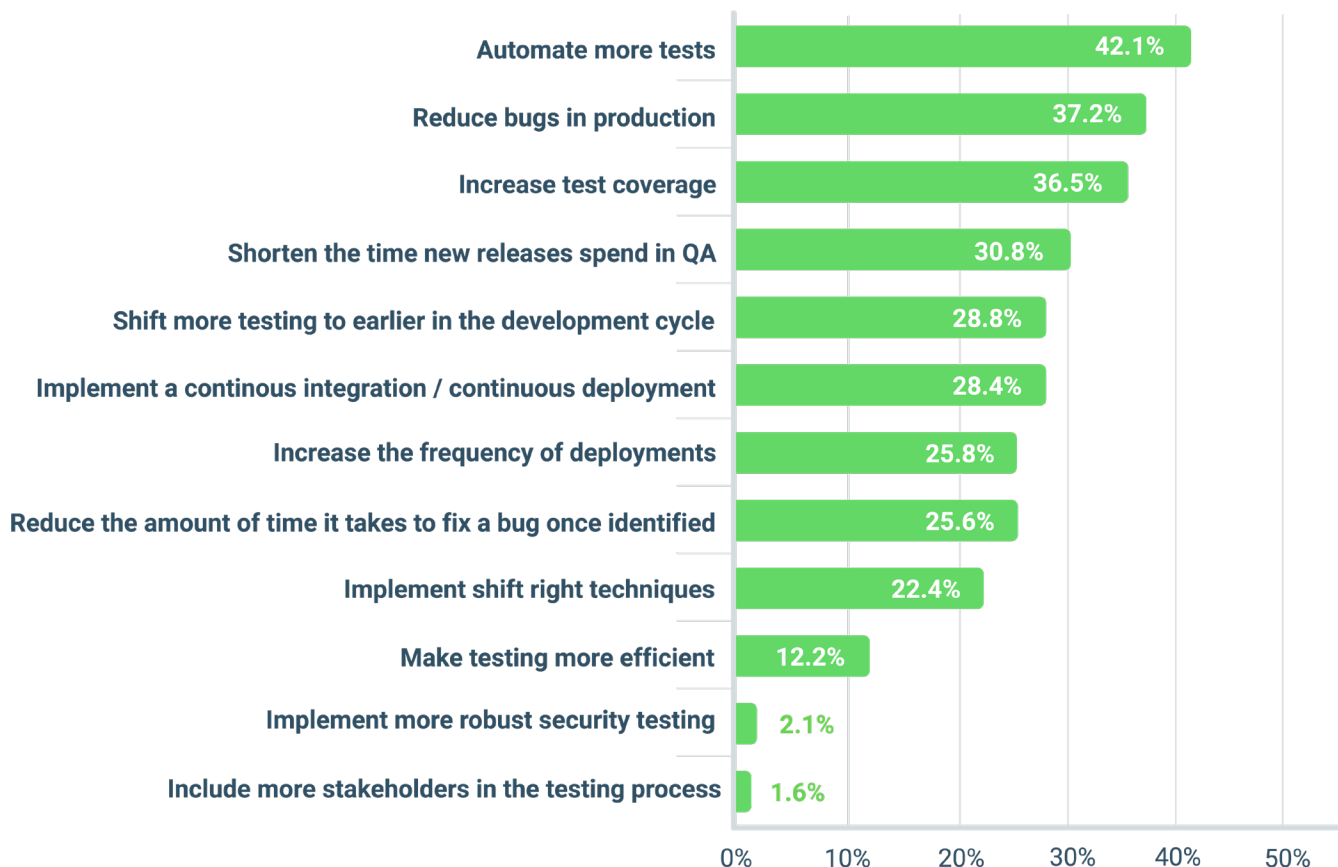
Earlier Software Testing and Quality Report data provides useful context on what can be gained from stronger integration:

- **86% of teams with high test automation and CI/CD integration report faster release cycles** compared to only 42% with low automation.
- **71% of teams with high test automation and CI/CD integration experience reduced defect leakage**, compared to 35% of low-automation teams.
- **58% of teams using automated test execution tied to CI/CD report a measurable return on investment (ROI)** from automation within six months.

These findings show that connecting test automation with CI/CD is not just a technical improvement. It accelerates feedback, identifies defects earlier, and increases the return on investment in automation. As AI accelerates code creation, fast feedback is more critical. Teams need automated tests integrated into delivery pipelines so they can validate changes earlier, detect risks faster, and avoid pushing more work into late-stage manual regression cycles.

What Teams Are Trying to Fix Next

What are your team's top 3 QA testing priorities over the next 12 months?



The 2026 Software Quality Pulse Report asked QA teams to identify their top 3 priorities for the next 12 months. Their answers point to automation, and the outcomes teams expect it to improve.

- ◆ **42.1% — Automate more tests**

This is the leading priority, consistent with the finding that 57% of QA tests are automated, leaving a substantial share still manual.

- ◆ **37.2% — Reduce bugs in production**

Escaped defects indicate that existing testing and coverage are not catching issues before release.

- ◆ **36.5% — Increase test coverage**

Teams want testing to cover more of the application, particularly the workflows and areas that pose the greatest risk.

- ◆ **30.8% — Shorten the time new releases spend in QA**

Teams are looking for ways to accelerate QA without sacrificing release confidence or coverage.

- ◆ **28.8% — Shift more testing left**

Earlier testing finds issues during development, before they become expensive and disruptive.

- ◆ **28.4% — Implement CI/CD**

This priority speaks to the integration gap, as only 26% of teams report strong QA/DevOps integration.

Other priorities include increasing deployment frequency (25.8%), reducing the time required to fix identified bugs (25.6%), and adopting shift-right techniques (22.4%). Together, these priorities describe teams working toward the same broad goals: catch more issues, catch them earlier, and resolve them faster.

AI adds urgency, and **60% of Pulse Report respondents** say AI-driven testing has improved QA coverage, quality, or speed, but only 17.2% describe its impact as significant. In other words, AI is contributing to improvement, but those gains remain uneven. At the same time, AI-generated code is increasing the volume of work QA teams must validate.

Teams know what they need to improve, but many of the same blockers persist year after year. Closing these gaps requires addressing their root causes, improving processes, and ensuring new tools are understood, documented, and supported with appropriate training before adoption.

Section 2

Why QA Teams Struggle to Automate More Testing

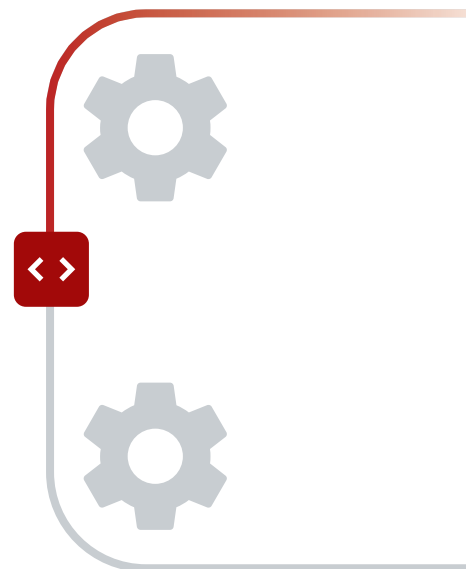
Despite growing pressure to automate testing, many QA teams face persistent roadblocks that prevent them from reaching their automation goals. These challenges are often rooted in team capabilities, technical complexities, and organizational dynamics.

Skill & Expertise Gaps

Limited access to coding and automation expertise makes it difficult to build and manage automation frameworks effectively. And the skills required go beyond writing automated tests: teams need expertise in design patterns, framework architecture, scalability, and selecting the right approach for specific testing needs.

Hiring skilled automation engineers is also difficult because demand outpaces supply. Even when companies invest in upskilling existing QA staff, teams must balance learning new skills with their day-to-day responsibilities.

This contributes to a [broader talent gap in the QA space](#), especially for roles requiring testing and programming expertise. It's also common for teams to lack the experience needed to design and implement a robust test automation strategy, leaving them without clear direction.



Applications Under Test (AUT) Complexity

Another factor is the growing complexity of applications. Distributed systems with cloud infrastructure, multi-tenant architectures, and microservices make testing far more intricate.

Tool limitations can compound this complexity, especially when they aren't fully evaluated before a framework is selected. You may adopt a tool only to discover that it can't automate critical workflows, support required browsers or platforms, or integrate effectively with your existing environment.

Tool Fragmentation & Lack of Centralized Management

Disjointed tooling across teams often results in fragmented automation. Different groups may use their own tools and processes, which leads to siloed automation that's hard to scale or manage centrally.

Integrating test data from manual and automated sources becomes difficult, hindering effective analysis and reporting. If you don't have a [centralized system](#) to manage and track test execution and results, inefficiencies and blind spots can clutter up your testing landscape.

This fragmentation creates several recurring problems that make it difficult to understand what has been tested, identify remaining risks, and demonstrate the value of automation.

- Manual and automated testing are tracked separately
- Test results are scattered across systems
- Different teams use different tools and standards
- QA leaders lack visibility into test outcomes, execution time, coverage, defects, root causes, risk, and ROI
- Automation work is duplicated across teams

Centralizing these metrics can also help teams identify environmental issues, understand where defects originate, compare regression findings with production defects, and determine where additional resources or process improvements are needed. Publishing this information gives stakeholders a clearer view of QA effectiveness and progress.

Time & Resource Constraints

Even when teams are committed to automation, time and resources are often limited. Automating complex scenarios can be labor-intensive, and in fast-paced release cycles, it's rarely prioritized over the immediate deliverables. Few teams have the bandwidth or dedicated resources to maintain and update automation scripts regularly. Over time, maintenance falls behind, test suites degrade, and automation progress stalls.

The hidden capacity loop

Manual testing consumes capacity. Limited capacity slows automation progress. Slow automation progress keeps manual testing high. Teams that don't deliberately break this loop tend to stay in it.

Test Stability & Flakiness Issues

[Unstable automated tests](#) are a common pain point. High maintenance costs and frequent test failures, often caused by UI changes or dynamically generated content, make it difficult for teams to rely on automation. Debugging false failures and false passes **takes up time and undermines confidence in the test.**

Inadequate Test Coverage Strategies

Another common issue is an unbalanced approach to [test coverage](#). Many teams concentrate too heavily on UI-level automation while neglecting other critical layers, such as APIs, integrations, and unit testing. End-to-end API tests can validate realistic workflows against live endpoints in a test environment, helping teams catch integration issues earlier and generate test data programmatically without relying on the UI. Without a [structured approach](#) to prioritizing what to automate, work is often misaligned with actual risk and impact, resulting in poor return on investment.

Lack of Executive Buy-In & Organizational Support

Automation often falters without strong organizational support. Leadership may remain skeptical about the return on investment when benefits such as time savings and increased reliability are not demonstrated. QA teams may also have limited input into tool selection or lack dedicated time for framework maintenance and technical debt, upskilling, and evaluating new tools through exploration and proof-of-concept testing. Without the resources and capacity to support this work, teams struggle to improve and scale their automation efforts.

AI Is Increasing the Cost of Staying Stuck

AI-assisted development can increase the speed and volume of code changes. For QA teams, that means more changes to validate, more regression risk, and greater pressure to provide fast feedback. If automation is already brittle, disconnected, or incomplete, AI amplifies those weaknesses.

At the same time, QA teams may be expected to use AI tools to accelerate their own work. These tools still require training, upskilling, and clear guidelines for appropriate use, review, and security. Without that foundation, AI can create additional work and introduce new quality and security risks rather than delivering the expected efficiency gains.

Section 3

How Mature QA Teams Close the Gap

If any of the challenges described above feel familiar, you're not alone. The test automation gap can be narrowed through careful, incremental improvements. Even small changes in one or more of the following areas can strengthen automation coverage, reliability, and efficiency.

Mature QA teams tend to improve automation through a progression that includes:

- Assessing current automation maturity and identifying the biggest gaps
- Building the skills and selecting the tools needed to address them
- Improving the reliability and maintainability of automated tests
- Connecting people, testing tools, and delivery pipelines
- Using AI where it can provide measurable value
- Measuring results and building the strategy and organizational support needed to scale and sustain automation

The sections below follow this general progression. You do not need to tackle every area at once, and you will probably find that you're further along in some areas than others.

Assess Your Current Automation Maturity

Before expanding automation, establish a clear picture of where your team is today. Look beyond the percentage of tests that are automated. Consider the reliability of existing tests, maintenance effort, coverage of high-risk workflows, integration with CI/CD, team skills, tool fit, and visibility into test results.

This baseline helps teams identify which constraints are actually limiting progress. One organization may need to stabilize a brittle test suite before adding coverage. At the same time, another may be held back by skill gaps, disconnected tools, or an automation framework that no longer fits its needs. Understanding those gaps makes it easier to prioritize improvements instead of trying to address everything at once.

[Take the Test Automation Maturity Gap Assessment](#)

Action Plan



Assess current automation coverage, reliability, maintenance effort, skills, tooling, and integrations.



Identify the gaps that create the greatest risk or slow the team down.



Establish baseline metrics so improvements can be measured over time.



Prioritize the next steps based on those findings.

Build Skills & Expertise

Training and certification programs provide structured ways to build skills and earn [valuable credentials](#). For newer testers especially, structured training can make a complex field more manageable by creating a repeatable cycle of learning, application, and improvement.

Supporting and prioritizing training shows that learning is part of the job, not an afterthought. Over time, this can change team culture. Knowledge sharing increases, conversations shift toward established patterns and practices, and clear growth paths make it easier to develop and retain talent. Training then evolves from a temporary response to skill gaps into infrastructure for continuous improvement.

Effective upskilling starts with the team's actual bottlenecks rather than a particular tool or technology. Give QA teams a role in identifying where they need support and defining the outcomes they want from training. Those needs may include low-code or full-code automation skills, but they may also involve test design, design patterns, framework organization, processes, assessment, or communication. Tool proficiency alone won't solve automation problems if teams lack the practices needed to structure, build, and maintain tests effectively.

Action Plan



Ask QA teams to identify bottlenecks and define learning goals based on the skills they need most.



Support training in technical and non-technical areas, from automation and test design to processes and communication.



Make regular learning part of the workflow, for example, one hour per week.



Create space for knowledge sharing through lightning talks, lunch-and-learns, or mentoring.



Incorporate skill development into performance reviews and career paths.

Choose the Right Tool for the Job

There are many different [automation tools on the market](#), but no single tool is right for every team. Tool selection should start with the people who will build and maintain the tests. QA teams can help identify the application requirements, testing challenges, and technical trade-offs that a tool or framework needs to address.

Before evaluating tools, understand where your current approach is falling short. Consider your application architecture, technology stack, automation maturity, team skills, testing requirements, and recurring pain points. Then evaluate whether prospective tools can support the workflows and environments you actually need to test.

Low-code tools can help testers contribute to automation sooner, while full-code frameworks offer greater flexibility for teams that need extensive customization. Neither approach is inherently better. The right choice depends on the application, the tests being automated, and the skills and resources available to build and maintain them.

- ◆ **Application and environment fit**

Match the tool to your application's complexity and workflows, plus the browsers, devices, platforms, and application types you need to cover.

- ◆ **Team skills and usability**

Weigh how well each option matches your team's existing skills and its overall programming model.

- ◆ **Maintainability and scalability**

Look at maturity, dependencies, long-term maintainability, and how the tool handles selectors and object management.

- ◆ **Integrations and workflow fit**

Check CI/CD and test management integrations to see how the tool slots into your existing pipeline.

- ◆ **Support, deployment, and governance**

Factor in documentation, community or vendor support, and any enterprise deployment requirements.

Action Plan



Assess pain points, test outcomes, recurring failures, and root causes before selecting new technology.



Have QA teams investigate candidate tools and frameworks, including capabilities, limitations, and trade-offs.



Build proof-of-concept tests for required workflows, browsers, platforms, and integrations.



Select tools based on demonstrated fit with your application, team, and testing needs.



Provide onboarding on tool usage, test design, framework patterns, troubleshooting, and supporting resources.

Build Reliable, Resilient Automated Tests

Accelerating test automation means little if the tests are flaky or brittle. Reliability should be a [design goal](#) from the start. Use explicit waits instead of arbitrary delays, isolate tests to avoid shared-state issues, and run them in clean environments. Organize tests by feature and purpose so teams can run targeted suites, including smoke or core tests that validate critical functionality earlier in development.

Layer the test suite intentionally. Unit and API tests can provide fast, targeted validation, while UI tests should focus on workflows that require end-to-end validation. Avoid repeatedly navigating through the UI when those steps aren't part of what a test is intended to validate. Where appropriate, use APIs, direct navigation, or other setup methods to put the application in the required state and keep tests focused.

Resilience is equally important. Use stable selectors and reusable patterns so minor interface changes don't require widespread test maintenance. QA and Development should agree on automation-specific attributes or other selectors that aren't changed without coordination, and communicate upcoming UI changes that may affect automated tests.

Tools can also reduce the work required to build maintainable automation. Depending on the approach, capabilities such as built-in object recognition, reusable components, and centralized object management can simplify patterns that engineers would otherwise implement and maintain within the framework. These capabilities support good test architecture rather than replace it.

Reliability is not a nice-to-have. It's the difference between the automation that teams trust and the automation they work around.

Action Plan

-  Organize tests by feature and purpose, with targeted smoke or core suites where appropriate.
-  Use APIs or other setup methods to bypass UI steps that aren't under test.
-  Use explicit waits, isolated tests, and clean test environments.
-  Establish stable selector conventions and communicate UI changes with QA.
-  Build reusable components and patterns that reduce maintenance.

Strengthen Cross-Team Collaboration

Test automation improves when QA collaborates with Product, Development, and DevOps throughout the development lifecycle. Product teams can share feature briefs and mockups early so QA can begin test design and automation planning before development is complete. Manual testers and automation engineers can also work together to combine product knowledge and exploratory thinking with automation expertise.

Collaboration should extend to test environments as well. QA and DevOps can work together to identify excessive latency, timeouts, unstable third-party services, and other environmental issues that can produce flaky tests and false failures. Separating genuine application defects from environment-related failures makes automation results easier to trust.

Mature teams treat quality as a shared responsibility, with testing information visible beyond the QA team. A shared dashboard can give technical and non-technical stakeholders visibility into manual and automated test results, regression findings, production defects, root causes, automation coverage, and the remaining automation backlog.

Action Plan

-  Share feature briefs, requirements, and mockups with QA early.
-  Establish shared conventions for stable automation selectors.
-  Pair testers and hold collaborative test-design sessions.
-  Work with DevOps to address test-environment issues that cause flaky tests and false failures.
-  Maintain a shared dashboard for test results, defects, root causes, automation progress, and backlog.
-  Document and share reusable testing patterns and lessons learned.

Connect Automation & Test Management

Test automation often lives in multiple places, with scripts on individual machines, results in CI dashboards, and documentation in spreadsheets. This fragmentation makes it difficult to see the complete testing picture. Centralizing testing through [test management tools](#) creates a shared home for manual and automated testing. Teams gain clearer visibility into what is working, what is covered, and what still requires attention.

When tools such as [Ranorex and TestRail are integrated](#), automated tests created in Ranorex can be linked to corresponding test cases in TestRail. Results then appear alongside manual tests, execution history, and test-run context, giving testers and stakeholders a more complete view of quality.

This integration addresses the fragmentation problem: manual and automated results in one place, a shared source of truth for coverage, and a unified view of risk.

Action Plan

-  Centralize manual and automated test cases and results in a unified platform.
-  Integrate tools such as Ranorex and TestRail to improve traceability and visibility.
-  Link test cases to user stories, requirements, and [CI/CD events](#).
-  Review coverage and execution data with the team regularly.

Use AI to Strengthen Test Automation

AI will not replace testers or instantly stabilize flaky tests. Its value lies in augmenting human effort, particularly in repetitive, time-consuming, or complex tasks. In test automation, AI can support test design, coverage analysis, and ongoing maintenance while leaving judgment and accountability with the testing team.

AI also cuts both ways. It can accelerate test design and maintenance, but it's also increasing the amount of software that QA teams must validate. AI-generated code requires the same scrutiny as other code, often under greater pressure to keep pace with a higher volume of changes.

Tools such as [DesignWise](#) can help teams design test cases by recommending combinations that reduce redundant testing while maintaining coverage of important interactions and paths. AI can also identify coverage gaps, automate parameterization, generate Gherkin test scripts, analyze patterns in flaky tests, and filter large volumes of test output to surface relevant information.

AI still comes with important caveats. Introducing it simply to speed up testing can create more work if you don't understand how the tools work, the risks involved, or the specific use cases they're meant to support. Generic models and poorly implemented workflows can also introduce silent failures when teams accept outputs without adequate review. Used thoughtfully, AI can expand what testers accomplish while keeping judgment, prioritization, and accountability with the testing team.

Action Plan

-  Use AI to support efficient test-case design and coverage analysis.
-  Analyze flakiness patterns to prioritize maintenance.
-  Train teams on AI capabilities, limitations, prompting, and appropriate use cases.
-  Establish security checks and code-review requirements for AI-generated outputs.
-  Require human review of AI-generated recommendations and changes.

Measure, Scale & Sustain Automation

A strong [automation strategy](#) starts with clarity. Define what you want to achieve, how you will measure success, and what resources and capabilities you will need. Assess how existing resources can be used more effectively, where processes need improvement, and how much capacity is needed to address maintenance and technical debt. Set realistic goals based on your current automation maturity, previous efforts, and organizational constraints.

Not every test should be automated. Use criteria such as execution frequency, business criticality, risk, and maintenance complexity to guide your decisions. Use the test automation pyramid as a flexible framework, with unit, API, and integration tests providing targeted validation and UI tests focused on critical end-to-end workflows. Where appropriate, automated checks can also be integrated earlier into development workflows, with smoke or core suites providing additional validation before changes progress to QA.

Securing organizational support is just as important as developing the plan. Involve the people who will build and maintain the automation when evaluating tools and defining the strategy. Make sure they have the training to use the selected tools effectively and agree that the approach fits the application, team capabilities, and available resources.

Measure the outcomes that show whether the strategy is working. Depending on your goals, these might include:

- ◆ **Coverage:** High-risk workflows and critical functionality
- ◆ **Reliability:** Flaky-test rate and false failures
- ◆ **Efficiency:** Execution time and manual testing hours
- ◆ **Quality outcomes:** Defect leakage and escaped defects
- ◆ **Sustainability:** Maintenance effort and technical debt
- ◆ **Confidence:** Release readiness and stakeholder trust

Action Plan

-  Use the test automation pyramid as a flexible guide, not a rigid target.
-  Establish baseline measurements and success criteria before scaling.
-  Account for maintenance, technical debt, and process improvements in the roadmap.
-  Assign clear ownership for test creation, maintenance, and reporting.
-  Select tools based on use case, maturity, capabilities, and available team resources.
-  Involve QA and other stakeholders early in the strategy and tool-selection process.

Section 4

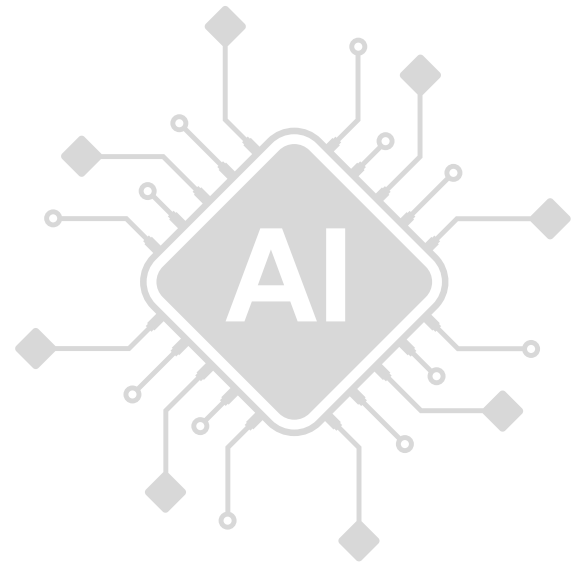
The Future of Test Automation in the Age of AI

Test automation is entering a more demanding phase. AI-assisted development can increase the speed and volume of code changes, creating more work for QA teams to validate. AI-generated test code can introduce another challenge when teams adopt scripts they don't fully understand, making failures harder to diagnose and automation more difficult to maintain. Keeping pace requires automation that can handle greater complexity, connect with development pipelines, and provide reliable feedback earlier in the software lifecycle.

AI-Driven Test Automation

AI is pulling testing in two directions at once: It's increasing demand for testing while also giving teams new ways to meet that demand. AI can support test design, identify coverage gaps, prioritize maintenance, analyze failures, and optimize test suites.

AI should not replace tester judgment or serve as a shortcut around smart testing practices. Its value lies in helping testers direct their time and expertise where it can make the most impact.



Shift-Left Testing Adoption

[Shift-left testing](#) gives teams earlier feedback by bringing QA into the design and coding phases of development. With feature descriptions and mockups, QA teams can begin designing manual and automated tests before development is complete. Shared selector conventions can also enable earlier automation preparation, leaving teams to validate and refine tests once the feature becomes available.

Automated tests integrated into [CI/CD pipelines](#) allow teams to validate changes instead of letting untested work accumulate until late-stage testing. Earlier validation becomes increasingly valuable as the pace of software development accelerates.

Tighter Integration Between DevOps & Test Automation

[DevOps integration](#) connects automated testing to delivery pipelines while providing developers, testers, and operations teams with shared visibility into test execution, coverage, and release risk.

Faster code changes require faster validation. Integrated automation allows teams to run targeted subsets, such as smoke tests, core tests, or tests for specific feature areas, providing faster feedback without running the entire suite every time. More frequent, focused test runs help identify issues earlier without introducing manual bottlenecks or leaving QA isolated at the end.

The future of test automation depends on connected, intelligent, and reliable systems that can keep pace with delivery.

Closing the Test Automation Gap

Even with today's advanced testing tools and methodologies, many QA teams still fall short of their automation goals. The causes are familiar: limited time, skill gaps, flaky tests, fragmented tools, and insufficient organizational support. These challenges slow progress, but they are not insurmountable.

AI has not created these challenges, but it is raising the cost of leaving them unresolved, turning a familiar test automation gap into something closer to a chasm.

Closing the gap starts with understanding where your team stands today: what is automated, what remains manual, where tests break, where tools are disconnected, and where visibility is limited. From there, you can stabilize existing automation, expand coverage, connect testing across tools, and measure results as you scale.

The right tools can support that progression. Ranorex helps teams create reliable UI automation across desktop, web, and mobile applications, while TestRail provides a centralized view of manual and automated testing. Together, [Ranorex and TestRail](#) help teams improve traceability, reduce fragmentation, and collaborate across the development lifecycle.

[Take the Test Automation Maturity Gap Assessment](#)

TestRail 

[Start a free trial](#)

[Watch the on-demand demo](#)

 Ranorex

[Start a free trial](#)

[Watch the on-demand demo](#)



About TestRail

Founded in 2004, TestRail provides test management software for development and QA teams worldwide.

TestRail helps QA teams build, connect, and optimize their testing processes. The platform centralizes manual and automated test management, bringing testing work together in one place and providing visibility across the quality operations. QA leaders can understand coverage, monitor execution status, identify risks, and build repeatable, scalable workflows without piecing together information from scattered tools and spreadsheets.

Through integrations with DevOps pipelines and testing tools, TestRail enables teams to share testing timelines, data, and insights across the organization. Teams can manage their work more flexibly, improve collaboration, and make better-informed decisions about software quality and release readiness.

Read the [TestRail customer success stories](#) or visit the TestRail pages on [G2](#) and [Capterra](#) for independently verified reviews.

TestRail helps teams move from fragmented testing processes toward faster, more confident releases.



About Ranorex

Since 2007, Ranorex has provided comprehensive UI testing tools for software teams working with difficult-to-automate applications.

Ranorex Studio provides all-in-one UI test automation across desktop, web, and mobile applications. Automation experts can use its full IDE, open APIs, intelligent code completion, refactoring, and debugging capabilities. Testers with less programming experience can use capture-and-replay functionality and built-in methodology to create reliable, maintainable tests while developing their automation skills.

Cross-functional teams can collaborate by sharing reusable object repositories and test automation modules. This combination of low-code and full-code capabilities allows people with different skill levels to contribute while giving experienced automation engineers the flexibility to customize complex tests.

Read the [Ranorex customer success stories](#) or visit the Ranorex pages on [G2](#) and [Capterra](#) for independently verified reviews.

Ranorex delivers robust, reliable UI automation for complex desktop, web, and mobile applications. Advanced object recognition and flexible low-code and full-code capabilities help mixed-skill teams create, maintain, and scale automation.