Qt Quality Assurance

WHITEPAPER

HANDBOOK

How to Get Started with Automated Testing

A **5-step guide** to implementing automated testing for your applications

Table of contents

Introduction

The Tale of the Bug That Came Back

Imagine this: You're on a tight release deadline and QA has just signed off on version 1.0. The next morning, your support team gets a flood of bug reports — some trivial, some showstoppers. Panic ensues. You wish you had caught them earlier, ideally before they ever shipped.

This is the moment many teams dread. We can no longer just rely on manual testing and hope for the best. We need automation. But this isn't as simple as turning on a tool. It requires discipline, planning, and the right mindset.

In this handbook, we'll walk you through 5 steps you can take to bring meaningful, lasting automation into your software process. Think of it as a travel map: you take it one step at a time, with milestones, guardrails, and stories from the field.

Who Is This Handbook For?

This handbook is designed for Implementation Leads, QA Managers, Software Engineers, DevOps Professionals, and Team Leads who are responsible for driving the transition from manual to automated testing within their organizations, as well as for optimizing their overall approach to test automation over time. It's especially relevant for teams working in complex application environments, from embedded systems to desktop, web, and mobile platforms, where quality assurance is critical and testing gaps can lead to costly regressions.

This handbook provides a structured, real-world roadmap to help you:

- Align cross-functional stakeholders on automation goals
- Assess team readiness and infrastructure compatibility
- Choose and integrate the right tools for your tech stack
- Build and optimize a maintainable test suite
- Measure impact and evolve your automation practice over time

If your role involves balancing delivery speed with reliability, and you're tasked with making automation sustainable, this guide is for you.

What Does **Successful Implementation** Look Like?

When organizations evaluate test automation solutions for their applications, they often focus on creating a comprehensive implementation plan that describes their goals and strategy for successfully adopting automated testing.

Implementation teams define objectives for each phase from initial assessment to full-scale automation deployment.

We've seen too many automation projects fizzle out. Why? Because success is not just "have tests that run automatically." True success means:

- **Measurable impact** — you can see how much time, bugs, or risk you've eliminated
- **Sustained adoption** — the team trusts and maintains the tests
- **Scalability** — your framework grows with new features and platforms
- **Integration with development flow** — tests run as part of CI, not as an afterthought

In other words: you don't just plan automation, you execute it. In our 5-step roadmap, we'll give you a framework for systematically approaching automation adoption, staying focused on measurable outcomes, and progressing methodically towards your testing goals.

Consider both technical training on automation tool capabilities and strategic training on automation best practices. **The Qt Academy course ↗** on automated testing provides structured learning that can supplement your internal training efforts.

Step 1
Setting the Foundation

You can't build a castle on shifting sand.

Assess the application & align expectations

When you begin test automation implementation, start with thorough assessment and preparation. Gather comprehensive information about your application architecture, testing requirements, and technical environment.

Understanding your application technology stack and testing landscape is crucial because different applications may require specific setup approaches. Gather your engineers, testers, product owners, and operations teams to ask:

- What are the most painful testing gaps today?
- What does "success" look like for this automation initiative?
- Which features or workflows are mission critical?
- What constraints (platforms, tech stacks, embedded hardware, etc.) do we have?

Don't skip this: if different stakeholders have different goals, your project will stall.

Evaluate team readiness & skills

Look at your team's mix of manual testers, developers, and QA engineers.

- Who will be involved in the automation efforts?
- Who knows scripting?
- Who knows CI/CD?

You'll need this knowledge to map out possible training to fill any gaps. Communication and planning are absolutely critical for successful automation implementation. Identify the training and knowledge transfer requirements for your team.

Your main goal in this foundational step is to ensure all stakeholders understand the automation objectives, technical requirements are clearly defined, and your team is prepared for the transformation ahead.

Infrastructure, tooling, and compatibility

Evaluate your current development and testing infrastructure. Understanding how your automation tools will integrate with existing workflows, continuous integration systems, and quality assurance practices allows you to plan integration points and identify compatibility requirements.

- How will the automation tool plug into your CI pipeline?
- Will it run headlessly (without rendering the UI) or faster in parallel?
- Do you have the appropriate hardware/infrastructure to run test environments?

Make sure all stakeholders are aligned on why you are doing this, what the risks are, and how you'll succeed.

Before diving into installation, it's important to choose a test automation tool that fits your specific needs.

Consider the technologies you need to support. Are you testing **Java applications ↗** **Windows desktop applications ↗** **Web apps ↗** **Android Automotive? ↗** **Qt applications ↗**

The right tool should align with your application landscape and long-term testing strategy. If you're comparing options, start by shortlisting a few strong candidates and exploring them with **free evaluation licenses ↗** before committing to a commercial investment.

Step 2
Installation and Setup

First, make sure your tools are friends with your code.

Following your chosen automation tool's **getting started documentation ↗**, install the automation software, configure licensing (evaluation or commercial), and establish the basic tool configuration for your application testing.

This setup process includes connecting to license servers for commercial installations and ensuring all team members have appropriate access to the testing tools.

Once your automation tool is installed, connect it to your application's UI. This means setting up element/object recognition so tests can "see" buttons, fields, windows, and more.

Use available **tutorials ↗** and documentation to ensure proper integration setup and avoid common integration challenges.

Integrate with your development environment

- Consider how test scripts will be stored, managed, and executed within your current development processes to prevent workflow disruptions and encourage team adoption.
- Decide where your test scripts live (a test repository or folder), how they're versioned, and how tests will be triggered from CI/CD. Ensure that your tests run in the same environments (OS, resolution, library versions) as your app.
- Configure your automation tool within your existing development environment and establish connections with your development tools, version control systems, and continuous integration infrastructure.
- Finally, create a few basic smoke or sanity tests, like launching the app, logging in or opening a menu, to validate that your test environment can talk to your application.

Step 3
Building Your Strategy

Don’t try to automate everything at once.
Be methodological.

Pick your first automation targets

You can begin with workflows that bring early wins.

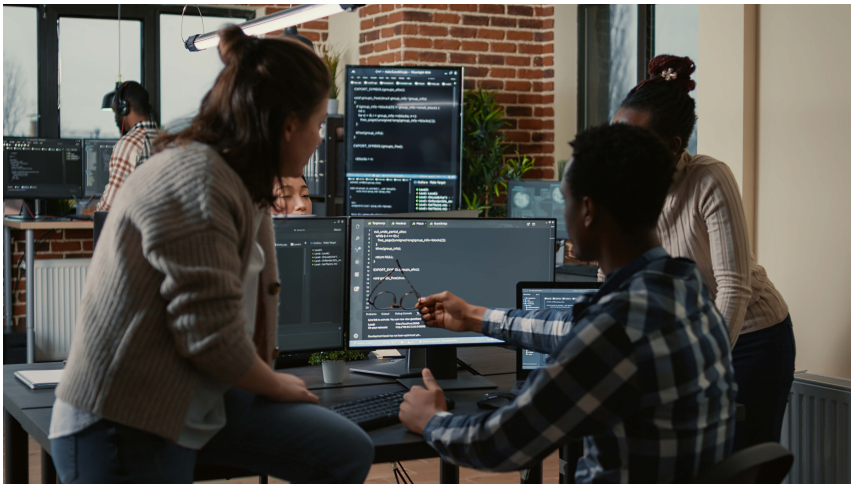
- Regression paths that you run often
- Critical user flows
- Complex UI interactions that are error-prone in manual testing

Define framework structure & coding discipline

Design the structural foundation for your test automation framework. This includes:

- Establishing coding standards
- Creating reusable test components
- Implementing data management approaches
- Defining reporting mechanisms

A well-designed framework architecture ensures that your automation efforts can scale effectively and that new team members can contribute productively to test development.



Establish and document automation best practices that align with your application requirements and team capabilities. These practices should cover:

- Test design principles
- Maintenance strategies
- Execution protocols
- Quality standards

By developing solid framework foundations and implementing proven best practices, you create a sustainable foundation for long-term automation success.

- Modularize tests (e.g. page objects, components)
- Define data-driven vs. scripted tests
- Establish reporting, logging, and error-handling conventions
- Integrate code coverage to track what's untested

Set guidelines: how to name tests, how to structure them, how to manage dependencies, how to update tests when UI changes.

Step 4
Expanding Coverage & Optimization

Scale with care, and the test suite won't become your enemy.

Grow test coverage incrementally

Once your proof-of-concept tests are in place, the automation journey truly begins. The next step is to **systematically expand your automated test coverage.** But, expansion should never be a race to automate everything at once. Instead, the smartest approach is to focus on where automation delivers the greatest return on investment.

This means identifying the test scenarios that matter most. Start with the ones that run frequently, such as regression suites, since they quickly pay back the effort of automation. Then, move on to checks that demand absolute consistency, like data validation, or to complex user interactions that are both time-consuming and error-prone when performed manually. These are the areas where automation can have the biggest impact.

As your coverage grows, so should your team's skills. True expansion goes beyond script writing, and centers on building automation maturity. Targeted training programs and hands-on practice ensure that the team can take on increasingly advanced testing requirements. Just as importantly, every team member should see how their role contributes to the broader automation strategy, so success feels collective rather than individual.

Knowledge sharing and collaboration become the glue that holds this growth together. As different team members solve problems or create reusable assets, encourage them to share lessons learned. This not only accelerates adoption but also prevents silos from forming as automation spreads across more features, edge cases, and platform variants.

Still, expansion must remain deliberate. Automating blindly often leads to bloated suites that are expensive to maintain and difficult to trust. Instead, keep ROI at the center of every decision. Try to prioritize:

- Tests that run daily or nightly as part of regression cycles.
- Scenarios that are especially difficult for humans to execute reliably, like complex UI flows, non-deterministic timing, or localization.
- Cases where human testers are prone to making errors under repetitive conditions.

By moving step by step (expanding thoughtfully, upskilling your team, and collaborating openly, you build a stronger, more resilient automation practice that grows in value over time.

Integrate with CI, monitor performance, and stabilize

Once your automation coverage begins to grow, the next step is to **integrate your test suite into development and continuous integration (CI) processes.** This shift ensures that automation provides immediate feedback to developers and becomes a natural part of everyday workflows, rather than an afterthought.

To make integration effective, consider three practical questions:

- How will automated tests be triggered?
- How will results be reported?
- How will test failures be triaged and resolved within the development process?

As integration deepens, attention turns to performance and reliability. A test suite that runs too slowly or fails often will quickly lose credibility. To avoid this, focus on:

- Improving execution speed
- Enhancing test stability
- Implementing robust error handling and clear reporting mechanisms

At this stage, it's also important to make test execution a visible, consistent part of your delivery pipeline. For example:

- Run smoke tests on every commit for immediate validation.
- Run full regression suites nightly or on each merge.
- Use parallel execution or distributed agents to accelerate feedback.

Finally, stability becomes the cornerstone of adoption. A flaky test suite erodes trust and discourages use, no matter how broad the coverage. Strengthen reliability by:

- Fine-tuning wait strategies, retries, and stable selectors.
- Reducing flakiness through continuous refinement.
- Monitoring key metrics such as time-per-test, failure rates, and stability trends.

By weaving automation into CI/CD pipelines and continually optimizing for speed and reliability, you ensure that your growing test suite remains both trusted and indispensable to the development process.

Step 5
Measure, Improve, and Plan for the Long-Term

Automation is never finished.

Focus on metrics & ROI

Once your automation foundation is in place, the next step is to measure its return on investment and define metrics for continuous improvement. Start by revisiting the objectives outlined in your initial strategy and assess how well your outcomes align with those goals.

Evaluate results across two dimensions:

- Quantitative metrics: test execution time, defect detection rates, coverage percentages
- Qualitative benefits: team satisfaction, workflow improvements, collaboration gains

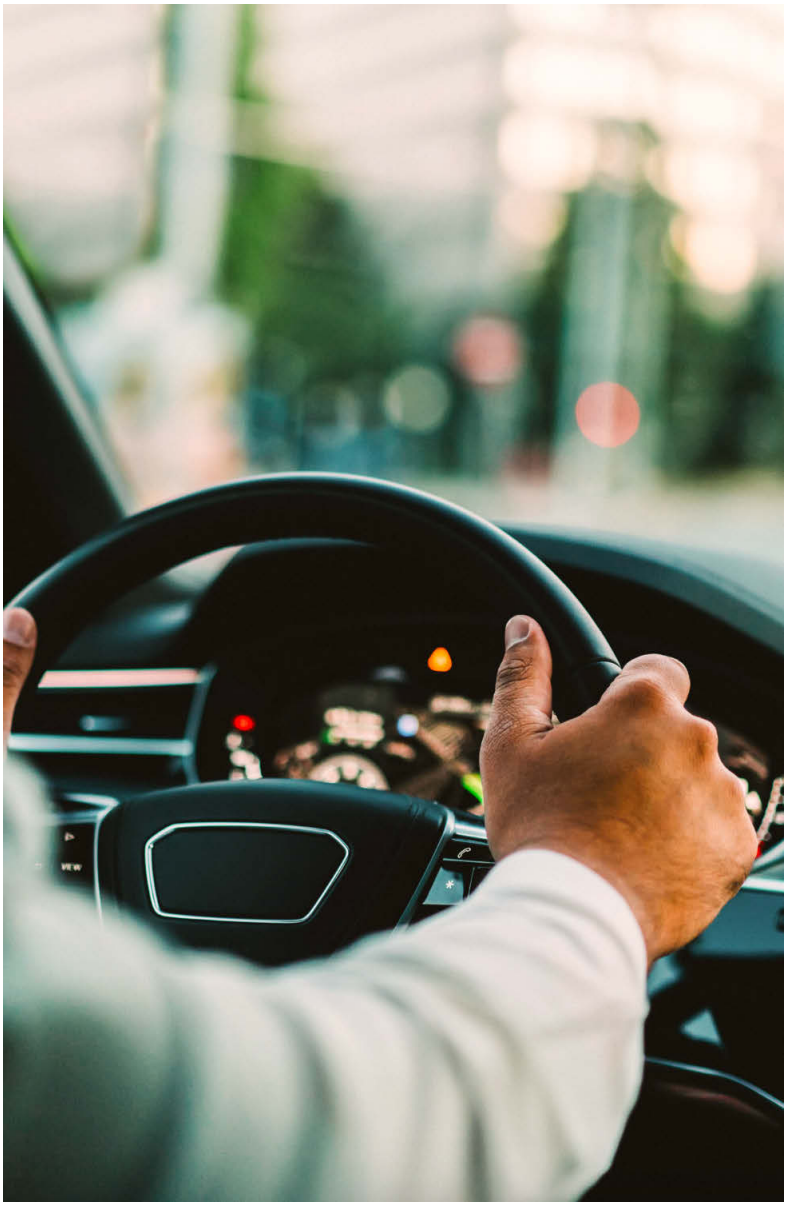
You can use the data to assess and document the measurable benefits achieved so far by asking:

- How much manual testing time you've saved
- How many regressions or bugs were caught earlier
- Test coverage growth
- Stability metrics like flake rates, test rejection rates
- Qualitative feedback: do teams trust automation?

To ground this analysis, compare your team's current productivity against baseline metrics captured at the beginning of your automation journey. Use a combination of execution data and direct team feedback to highlight successes and uncover opportunities for refinement.

With this foundation in place, you're ready to expand and optimize.

- Extend coverage into new modules or applications.
- Improve cross-platform re-use for greater efficiency.
- Adopt advanced techniques, such as data-driven testing, AI-assisted test creation, or visual validation.
- Revisit your framework to prune underused tests, reorganize structures, and maintain scalability.
- Share successes and lessons across teams to accelerate organizational maturity.

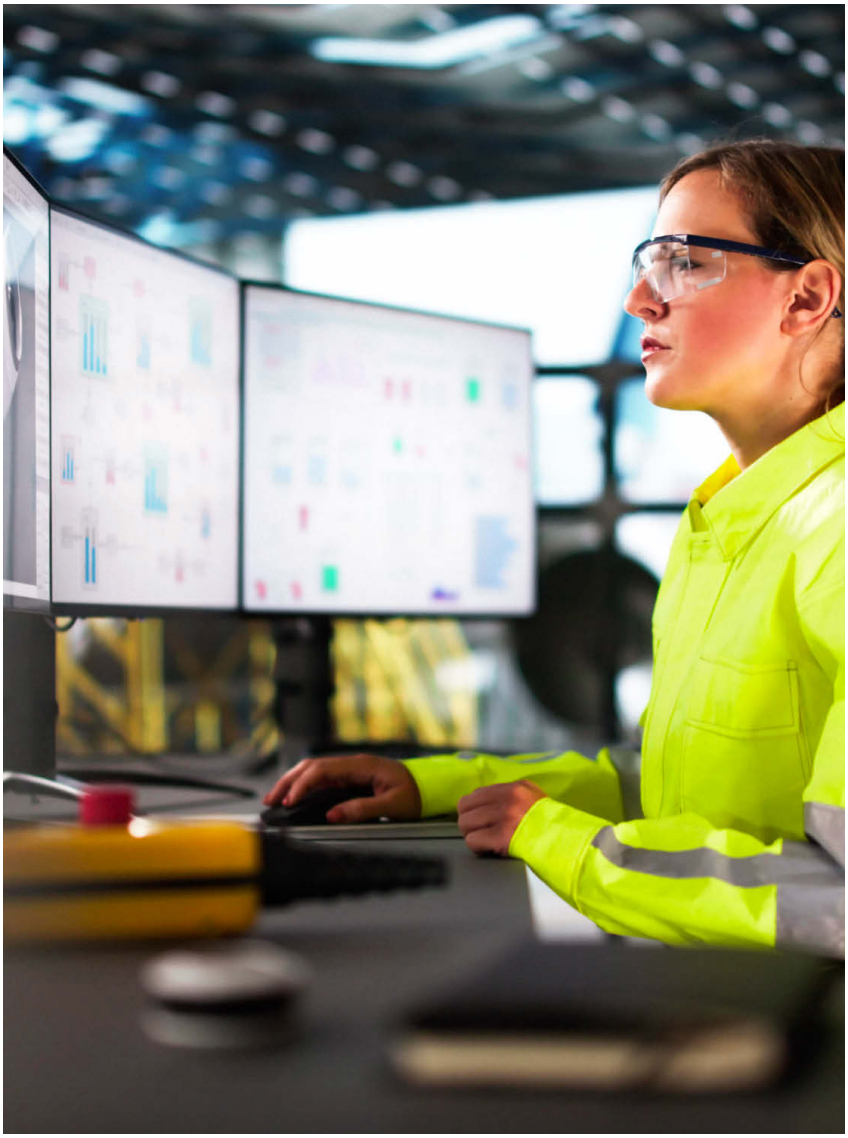


Summary & Takeaways

If you are shifting from manual to automated testing, it's tempting to sprint head-first. However, the strongest automation is built step by step. As an implementation lead, your role is to navigate not just the technical journey, but the organizational changes.

1. **Set the foundation** — align goals, assess readiness
2. **Install & configure** — get tools talking to your app
3. **Strategy & initial tests** — pick the low-hanging fruit
4. **Scale & optimize** — expand smart, stabilize the suite
5. **Measure & evolve** — keep improving, scale the practice

By following this structured path, and always keeping the human factors (trust, maintainability, clarity) in mind, you will dramatically increase your chances of real, lasting automation success.



Visit the Software Quality Solutions Page ↗

Follow the Software Quality News newsletter on LinkedIn to get early access to the latest trends in Software Quality Testing ↗

Explore more resources on automated testing through our whitepapers and webinars ↗

Explore free evaluation licenses ↗

Your Next Move

Are you ready to bring this to life in your organization? Here's how to get started:

- Choose one small, high-value workflow and apply **Steps 1–3** just for that scope.
- **Shortlist testing tools that align with your project's requirements, environments, and long-term goals.** Begin with evaluation licenses or free trials so your team can experiment in real conditions without heavy upfront investment. This hands-on exploration will quickly reveal which tools fit best into your workflows.

For example, within the Software Assurance Solutions by Qt Group:

Squish GUI Tester ↗ is ideal if you need robust cross-platform GUI testing that works across desktop, web, mobile, and embedded applications.

Coco Code Coverage ↗ provides advanced code coverage analysis for both desktop and embedded systems, helping you understand which parts of your codebase remain untested.

Test Center ↗ gives you centralized visibility into test results, making it easier to manage, analyze, and share testing outcomes across teams.

Axivion, Static Code Analysis and Architecture Verification ↗ helps you prevent code smells, enforce design rules, and keep technical debt under control.

By starting small with trials and proof-of-concepts, you will not only validate technical fit, but also secure stakeholder buy-in before scaling automation across your entire organization.

- Book a session with your team, where you walk through the 5-step plan and get buy-in.
- Monitor early metrics and iterate quickly.

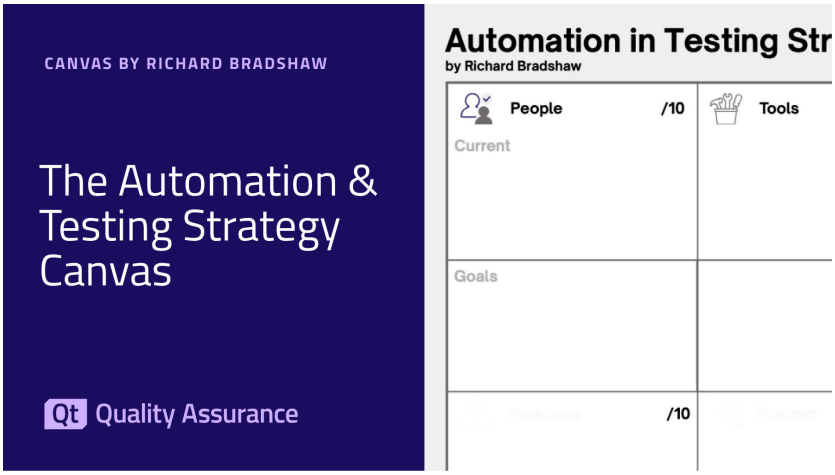
This step-by-step approach ensures methodical progress toward automation excellence while maintaining focus on measurable business value and sustainable practices.

Good luck on your test automation journey, and remember, you don't have to walk it alone! On the next page, you'll learn how to put the strategies you've learned into action.

Start Your Planning with The Automation & Testing Strategy Canvas

Once you've explored this handbook and chosen the tools, the next step is turning insight into action. That's where the Automation & Testing Strategy Canvas by Richard Bradshaw comes in.

This simple visual framework will help your team bring clarity, alignment, and focus to your automation approach. Use the canvas in team discussions, planning sessions, roadmap conversations, or retros to make your automation strategy visible, intentional, and collaborative.



This simple visual framework helps your team bring clarity, alignment, and focus to your automation approach.

The canvas will help you cut through that noise, giving you a shared language and a structured way to think, so you can clearly define:

- Where your team is today
- Where you want to be
- And the path to get there

Learn how to run a workshop with The Automation & Testing Strategy Canvas and get your PDF copy of the canvas