TestRail

# Trigger Your Test Automation Strategy:

## The QA Leader's Guide to Effective Test Automation

TestRail

# Index:

# Introduction

If you're on a software QA team these days, you're probably running some kind of test automation. Automation is a vital component of any modern testing program—our 2023 Software Testing & Quality Report found that 62% of respondents ran more than 100 automated tests per day in 2022, and we've found that QA teams are consistently automating about 40% of their tests year-over-year since 2019.

It's easy to see why test automation is so important for successful QA. Automating tests allows you to test more features faster, get releases out the door sooner and with fewer defects, decrease human error, test at scale, and free up your team's time and resources.

However, developing an effective test automation program that produces tangible results is often a bigger challenge than QA teams bargain for. As you launch and scale your automation program, you'll likely find friction around deciding what to automate, surfacing results from automated testing, the costs of maintaining your test automation infrastructure, adopting tools and framework that your team can support, and integrating automation into your agile test cycles.

While automation is key to any modern testing team, few teams build out their automation with a way that allows them to achieve consistent success. This guide is intended to introduce key concepts around test automation, steps to guide your automation strategy, and help you measure the success of your automation to deliver consistent results.

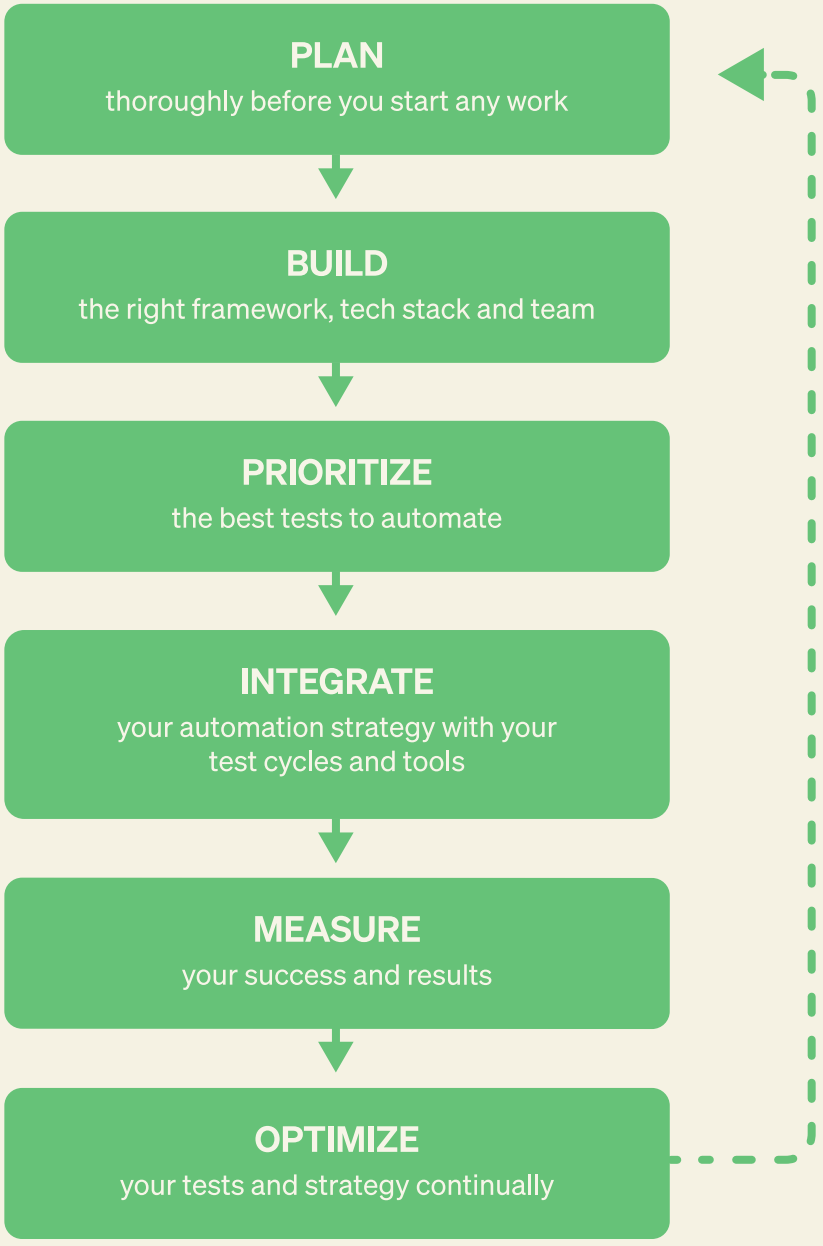
Trigger Your Test Automation Strategy

# Expanding the Role of Test Automation in Your Testing Strategy

# Expanding the Role of Test Automation in Your Testing Strategy

When it comes to forming a test automation strategy, many organizations aim to automate as many of their existing manual tests as possible. If you can simply take what you're already doing and automate it you're all set, right? Not necessarily—this approach will certainly increase your number of automated tests, but it may not give you the efficiency and improvements that will earn a good return on your investment.

Building and perfecting your test automation strategy might seem like a daunting prospect, but it can really be distilled down into six concrete steps:



Trigger Your Test Automation Strategy

# Implementing and Balancing Automation

# Implementing and Balancing Automation

**Five questions to ask before starting with test automation:**

## 1. What is our team's goal for test automation?

The answer to this question forms the basis for all decisions made after. A unified guiding goal, a north star, will provide the direction your team needs when undertaking such a significant project. This goal also forms the metrics for how you will define success—at this point of the process, you should be deciding what you will be quantifiably measuring to ensure that your automation strategy is helping your team achieve this goal.

## 2. How are we going to implement test automation?

Implementing and maintaining automation isn't just a side task. Different teams and skillsets are often responsible for the many moving parts that go into a successful automation suite, and you may need to loop in talent from outside of your current testing lineup. Decide on your allocation of talent first and then select tools and technologies based on their expertise as well as the needs of your system.

## 3. What is our execution strategy?

Your execution strategy should answer all of the questions you anticipate your team could ask regarding the day-to-day running of the automation program. This includes questions like:

- When would you like to run your automated scripts?
- At what level are we running these tests? Component, integration, end-to-end?
- Where are we running them—do we have adequate environments?
- Where are we running these tests? I.e., at what level and against what environment?
- How frequently are they supposed to run?
- Who will be running them—or would you like an automated trigger as soon as any new code gets checked in?

Depending on your answers to these questions, you may need to look at the continuous integration (CI) system you already have (or need), and any additional setups or plugins required to run your automation scripts.

# Implementing and Balancing Automation

## 4. Who will focus on maintenance?

Automation is unfortunately not "set-it-and-forget-it." Your automation program will require consistent test script maintenance and troubleshooting, in addition to adding and removing tests as needed. Everyone on an agile team should feel ownership over automation work and be empowered to maintain and improve it, however, we also recommend defining a test automation champion on your team to keep an eye on your automation efforts and surface alerts and concerns to the rest of the team.

## 5. How will we report on test automation?

Go back to step one. Remember how we asked you to define your team's goal and how you'll measure your progress towards it? Now, it's time to develop the systems and processes necessary to reliably and quantitatively report on those goals, at set intervals that make sense for your team's Software Development Life Cycle (SDLC).

Once you start to get metrics and results back from your automation efforts, you should get a sense for how automation is fitting into your holistic testing strategy. It's important to choose tools that allow you to oversee, measure, and report on manual and automated testing in one place—and structure your reporting process so that both manual and automated tests can be included in test coverage and traceability reports.

It's critical to also track and report on time: how fast are automated tests being executed? Are there opportunities to improve automated test runs to be more efficient with their time? We'll focus more on that in the "Which Tests Should be Automated" chapter below.

## "Automate everything" is not the answer

If you just "automate everything," you're not doing any evaluating. It may feel like you're saving time and resources in getting your test automation strategy off the ground, but you'll end up with a bunch of automated tests with no clear purpose or goals. And automated tests still have resource costs—human developers will need to maintain and troubleshoot those tests, and there are time and server costs to run them and store results.

Additionally, not every test is even a good candidate for automation. Automation is best for testing "happy paths" of expected user behavior, but it can't replicate unexpected user behavior to find edge cases. No testing method is better at performing odd human behavior than human testers—by removing the human element from your testing program entirely, you open yourself and your application up to more risks.

# Implementing and Balancing Automation

## Automation is not built in a day

Even with the best-laid plans, a perfect automation program is not going to spring forth overnight. The first step of building your automation strategy should be getting it to “good enough,” launching, and setting realistic goals to improve and increase over time.

The most worthwhile investment you can make in the beginning is on your automation infrastructure. If you know that your infrastructure is solid, then when something fails, you’ll be able to hone in on the failure of that thing specifically rather than having to examine your entire automation program.

Trigger Your Test Automation Strategy

# Select the Right Automation Framework and Tools

# Select the Right Automation Framework and Tools

Not all test automation frameworks and tools are created equal—they can vary widely in costs, strengths, capabilities, and ability to scale and customize. Some are “low code” and some require proficient technical knowledge. So which is the best choice for your team? You can use some parameters to help guide this decision:

## Who will be using the framework and conducting the tests?

The skills and strengths of the team members involved should be a major factor in your decision. While of course people can learn new skills, if your framework requires JavaScript and nobody who will be working with it already knows JavaScript, that’s not the kind of thing that can be learned overnight. Also take into consideration the learning curve required to learn the framework and how that will impact onboarding.

Also consider who will be conducting the tests—if you plan on having your manual testing team also manage automated tests, maybe a low-code option would be best. If you’ll be having developers write their own test code, pick a tool that uses the same language they’re already familiar with. If you have specialized test automation engineers, consult them on their recommendations and experiences. And if everyone will be involved, you need to come to an agreement on what is best given your project’s context.

## Who will be using the framework and conducting the tests?

Not every framework will be the best choice for every strategic need. Ensure you consider these requirements when comparing automation frameworks:

- Technologies being tested (mobile app, web app, etc)
- Test levels (component testing, end-to-end testing, etc)
- Limitations (such as cross-browser testing)
- Integrations with other tools (such as your test management platform)
- Customization (do you have the ability to add/create plugins for everything you need?)
- Reporting (how easy will it be to show results and debug when tests start failing?)

# Select the Right Automation Framework and Tools

## What are the general framework quality indicators?

Even if a framework looks perfect on paper, you should also take into account some general quality indicators.

### Support and Usability

- Does it have robust documentation?
- Does it seem to want to help you develop tests in an easier manner?

### Maintenance and updates

- When's the last time the framework was updated?
- How often is the framework updated?
- Does it appear to be well-maintained?

### Community size and support

- Is there a large, active user community?
- Is there a place (such as a community forum) where you can go to seek community support if you need help?

## Does your proof of concept make sense?

Before committing to a framework, you should always do a proof of concept using that framework in your actual project. This will ensure that it works in context, with all the technologies you need to test.

As part of this, be sure to involve everyone who will be developing the test cases in this proof of concept. If you don't, you might end up with a framework that works for you but not the rest of your team—which will lead to nothing but problems in the future.

Trigger Your Test Automation Strategy

# Which Tests Should You Automate?

# Which Tests Should You Automate?

After you assess your testing strategy, prune out unneeded or redundant tests, and maybe even add some new ones, you'll have a stack of tests and the million dollar question—to automate or not to automate? Start by asking yourself these questions.

## Questions to help you prioritize what to automate next:

### Is the test going to be repeated?

Automating a test case will be worthwhile if it is going to be repeated multiple times during test execution. Make sure to consider the frequency with which you plan to execute that test case or the test suite.

### Is it a high-priority feature?

Some features or areas are more prone to failure than others, and those are the areas that will be a bigger return on your investment in automation. Automating high-priority tests will help you identify more significant risks earlier in the testing process and reduce the chances of human error.

### Do you need to run the test with multiple datasets or paths?

A data-driven approach to test automation can be the most useful framework when tests need to be executed with varying datasets. Automating once and repeating the same steps with different data takes away the drudgery and minimizes the chances of misses or errors.

### Is it a Regression or Smoke Test?

Regression tests and smoke tests are the ones you will end up executing the most frequently. Generally, these test suites are the ones that cover the width of the entire product in some capacity, so automating them will give you a quicker way of assessing the quality of the entire software. Automating regression test suites can help integrate them with the build process in your DevOps pipeline thus making quality a part of your regular build!

# Which Tests Should You Automate?

## Does this automation lie within the feasibility of your chosen test automation tool?

Doing a feasibility analysis of your test automation also requires you to see if it is even possible to perform that specific test automation using your current set of tools. For example, trying to automate inputs to a mobile app for one specific test might not be the best use of your time if it isn't supported by your web automation test tool. Acquiring another tool or trying alternates might not be a good use of the time either. You might very well perform that one test manually in such a case!

## Is the area of your app that this is testing prone to change?

This is an important question to ask yourself because If the test you are considering automating is prone to change, it might not be worthwhile to spend your effort on that just yet. Moreover, automated tests that rely on the user interface and the UI elements are the most prone to break when the UI changes.

Take a deep look at the upcoming changes in the functionality and its integrations with related features, if you see big changes ahead, perhaps delay the automation of those tests for now.

## Can these tests be executed in parallel, or only in sequential order?

If you design your automation well, you can save an immense amount of time by running tests in parallel. Leveraging this is paramount to getting the best out of your automation efforts. If your tests are only executed in a certain sequential order, they might not be the best candidates for automation. Although you could still automate them, you would not be making the best use of your efforts.

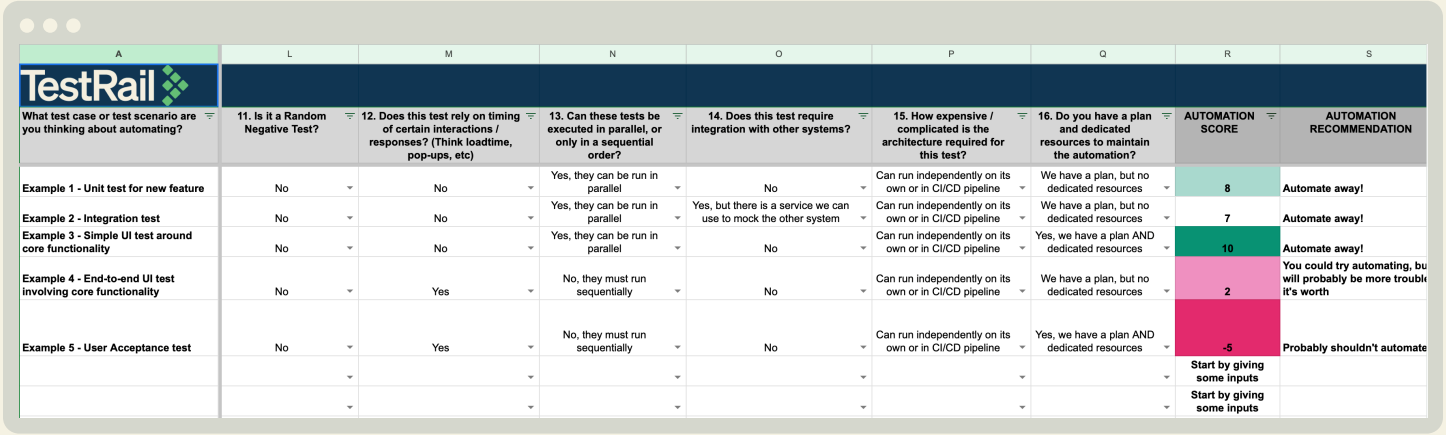
# Which Tests Should You Automate?

## Scoring model:

## To automate or not to automate?

That is the question. To make this decision easier on you, we've developed this scoring model to help you prioritize what to automate next and guide your automation strategy. Download this interactive, customizable spreadsheet here:

**Download Now**



| A | L | M | N | O | P | Q | R | S |
|---|---|---|---|---|---|---|---|---|
| TestRail | | | | | | | | |
| What test case or test scenario are you thinking about automating? | 11. Is it a Random Negative Test? | 12. Does this test rely on timing of certain interactions / responses? (Think loadtime, pop-ups, etc) | 13. Can these tests be executed in parallel, or only in a sequential order? | 14. Does this test require integration with other systems? | 15. How expensive / complicated is the architecture required for this test? | 16. Do you have a plan and dedicated resources to maintain the automation? | AUTOMATION SCORE | AUTOMATION RECOMMENDATION |
| Example 1 - Unit test for new feature | No | No | Yes, they can be run in parallel | No | Can run independently on its own or in CI/CD pipeline | We have a plan, but no dedicated resources | 8 | Automate away! |
| Example 2 - Integration test | No | No | Yes, they can be run in parallel | Yes, but there is a service we can use to mock the other system | Can run independently on its own or in CI/CD pipeline | We have a plan, but no dedicated resources | 7 | Automate away! |
| Example 3 - Simple UI test around core functionality | No | No | Yes, they can be run in parallel | No | Can run independently on its own or in CI/CD pipeline | Yes, we have a plan AND dedicated resources | 10 | Automate away! |
| Example 4 - End-to-end UI test involving core functionality | No | Yes | No, they must run sequentially | No | Can run independently on its own or in CI/CD pipeline | We have a plan, but no dedicated resources | 2 | You could try automating, bu will probably be more troubl it's worth |
| Example 5 - User Acceptance test | No | Yes | No, they must run sequentially | No | Can run independently on its own or in CI/CD pipeline | Yes, we have a plan AND dedicated resources | -5 | Probably shouldn't automate |
| | | | | | | | Start by giving some inputs | |
| | | | | | | | Start by giving some inputs | |

Trigger Your Test Automation Strategy

# How to Integrate Automation with your Test  Cycles

# How to Integrate Automation with your Test Cycles

In order for automation to make your testing truly more efficient, it needs to integrate intelligently with your test cycles. If your test cycles are not synchronized with your development cycles—or you segregate your automation team from your development team altogether—that's not agile and that won't help you achieve faster, more efficient releases.

## Decide where automation efforts will fall across your sprints

Teams generally approach automation efforts within their sprints one of two ways—executing development and test automation activities simultaneously within the same sprint, or alternating efforts, so features developed in one sprint are automated in the next one.

Having development and automation efforts running in parallel certainly gets your tests automated faster, but it takes a full-team effort to execute efficiently. If you only have one automation tester for a whole team of developers, you won't be able to automate everything–especially features developed late in the sprint. The whole team will need to be on board for both developing features and then automating the tests for those features for this approach to generate benefits.

Alternating development and automation efforts allows your team to fully concentrate on one thing at a time, ensuring steady progress with both development and test automation even if you don't have a large team able to dedicate their time to both during each sprint. However, this method means that your development and automation efforts will always be on a one-sprint delay—and your team must manually test features that are developed within the "development" sprints as they won't be automated until the next sprint.

The most important principle of test automation in an agile workflow is ensuring that your whole team is working together towards your goals. If you segregate your automation team from development, that's not agile. However, you can loop in an automation support team when additional resources are needed, such as when you are creating your automation framework.

## Establish a process for your testing workflow

Whether you automate your tests within the same sprint or on alternating sprints, you need to establish a process for your testing workflow and ensure the rest of your team is on board.

After features are developed, you should then:

1. Design test cases
2. Execute test cases manually to validate that sprint's features
3. Automate tests only after the features are validated (either within the same sprint or on the next)

# How to Integrate Automation with your Test Cycles

## Determine how you will execute automated tests

After your tests have been automated, you need to establish a strategy for executing them. Teams generally execute their automated tests one of three ways:

### On every deploy or merge request

- Merge requests happen very frequently, so these tests would have to run fast
- However, this method would provide the fast feedback needed for a “shift left” approach
- Deploys happen less often than merge requests

### On a trigger

- Tests are run on-demand—someone manually triggers the testing

### Overnight

- Tests are run every night, overnight

## Additionally, you’ll want to strategize how you batch and organize your automation execution.

- ✓ Batch and prioritize test cases into a series of executions that get progressively more advanced (such as smoke tests to regression testing)
- ✓ This way, if a smoke test fails, you only have to re-run that one quick execution rather than a full regression

- ✗ Don’t batch all of your automated tests into one huge execution
- ✗ If a single high-priority test fails, then that entire execution will have to be run again

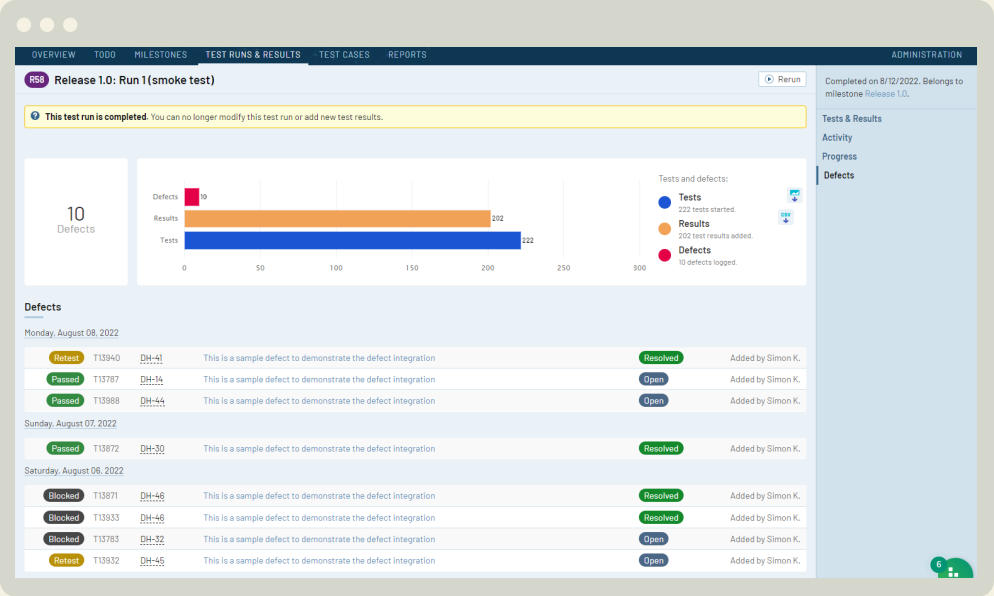
Trigger Your Test Automation Strategy

# Integrate Automation with your Test Management Tool

# Integrate Automation with your Test Management Tool

It's important to **integrate your test automation with the rest of your test management process.** One of the challenges teams face around test automation, especially as they get started, is a lack of visibility. By integrating your test automation with your test management tool, you can make it easier for your team to get access to your automation results, measure the output of test automation more easily, and take action on any product risks they identify.

When selecting a test automation tool or framework, make sure it can integrate with your test management platform. This functionality is commonly available via a command line interface (CLI) tool, API connection, or plugin. Once integrated, automated test results are automatically sent to your test management platform as tests are completed. This allows you to view your automated and manual testing data all in one place, giving you a holistic view of your testing program. The automatic transfer of test results lets you see defects and risks as they surface, empowering your team to take action faster. Additionally, such an integration also allows you to run and analyze reports on automated test data using the full capabilities of your test management platform.




## How to Measure the Success of Your Automation Program

You've prioritized what to automate, built your infrastructure, and officially launched your automation program. What now?

Now is time for the most vital step of all: measuring and reporting on your results. This will help prove your automation program's ROI, identify areas for development and resource allocation, and steer your strategy for the next update and beyond.

# Integrate Automation with your Test Management Tool

## Prioritize reporting and analytics

Just integrating your automation with your test management tool is not enough—you must also regularly assess the health and success of your testing program via testing data, and provide actionable and easy-to-parse reports to your stakeholders and team.

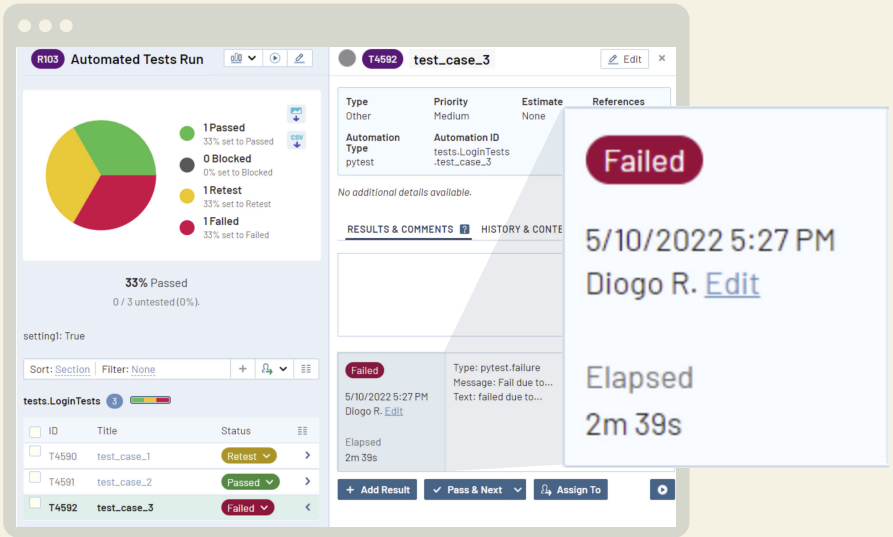
You're likely already doing quite a lot of analyzing and reporting with your existing test program, and the good news is that test automation doesn't require you to reinvent the wheel. In fact, segregating your manual and automated test data will do you quite a disservice. Your test automation data should be included alongside your manual testing data when performing analysis and reporting so that you and your team can see the full picture of how your testing program is performing.

A good test management platform should make it easy to pull reports on your cases, defects, and test results—and roll that data into summary reports that paint a clear picture of the health of your milestones, test plans, test runs, and projects as a whole. These reports should include all testing data, both manual and automated.

Additionally, by integrating your test automation results into a test management tool that also integrates with an issue tracker such as Jira, you can roll your manual and automated tests into traceability and test coverage reports. These reports ensure that all product functionality is being tested, and establishes a documentable trail between the work being done on your QA and development teams.

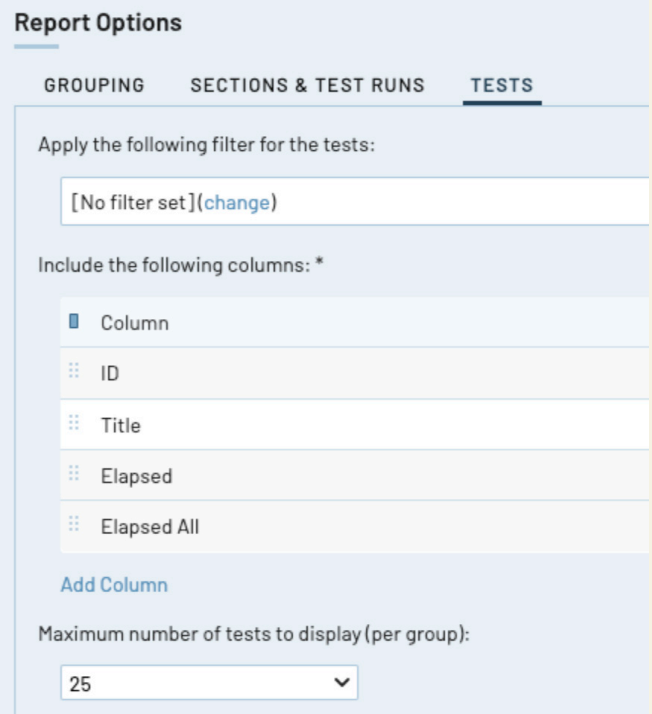
A few automation-specific items that you should be sure to include in your reporting are execution time and test flakiness.

Using TestRail as an example—test execution time is one of the data fields that will be populated when you integrate your automation platform using the TestRail CLI. [You can see an example of test details showing the elapsed time it took to execute the test here:]



# Integrate Automation with your Test Management Tool

Tracking test execution time gives you visibility into global execution time trends, as well as allowing you to identify tests that are especially time consuming so you can optimize them. You can view execution time via a TestRail report by producing a Summary - Runs report and adding 'Elapsed' and 'Elapsed All' to the Tests section while setting it up.




**Test flakiness**—a term used to describe tests that produce different results through multiple executions without any change to the script or system under test—can also be easily tracked in a test case management tool. In TestRail, you can accomplish this by adding a custom result field named "Flakiness" that includes the status values of:

- **Not Flaky**
- **Flaky**
- **Investigate**

As automated tests fail and testers explore the cause, they can assign a status value to flag tests that they identify as flaky or in need of further investigation. You can then run a report that groups tests by Flakiness to quickly surface tests that need design correction—allowing you to quarantine them from your test suite before they create further risk of defects.

You can learn more about flagging flaky tests in the TestRail Academy.

# Integrate Automation with your Test Management Tool

## Ensure visibility across teams and stakeholders

Now that you're measuring your success, make sure you're sharing it! Ensure your QA team has access to the latest testing data and reports so they can see the impact of their work in real time. The dashboard of your test management tool is a great place to share up-to-the-minute results, and consider also sharing insights and learnings from your analysis during your standups or team meetings.

You should also be sharing regular summary reports with stakeholders external to your team to keep them informed on your high-level insights and progress. Be sure not to overburden external stakeholders with too much information—curate your summary reports in your test management tool to only generate the information that each stakeholder needs to know.
While your manual and automated test data will be in one place, be sure to keep an eye out for trends and shifts that will come after introducing new automation. You should have plenty of testing data to use as a benchmark before introducing automation—what changed after you launched or made updates to your automation program? Hopefully the data will trend towards faster, more efficient releases—but keeping a careful eye on the impact of new tests and programs can also highlight areas you need to improve or reconsider.

The greater the visibility your QA team, development team, and external stakeholders have into your testing program, the easier it will be to be truly Agile. Visibility is a good practice for any testing program, whether manual, automated, or any combination thereof.

Trigger Your Test Automation Strategy

# Conclusion

# Conclusion

The foundation of any successful automation program is a solid strategy. We hope this guide helped provide you with a better understanding of how to approach automation for your team's unique needs, integrate automation into your existing testing program, and define and measure its success.

If you're excited to get started, the best place to begin is with a solid test management platform. Look for an option that allows you to centralize manual and automated testing data as well as activity from all your testing, tracking, and project management tools in one place—such as TestRail.

If you'd like to try TestRail out for yourself, we have two ways to get started:

**Sign up for our weekly TestRail product demo webinar**

Sign up

**Try a 14-day free trial of TestRail**

Download Now

You can learn more about TestRail at testrail.com. You can also visit our support center and review our test automation documentation here.

# Acknowledgements

This guide was compiled by Jeslyn Stiles, TestRail Senior Content Marketing Manager, and was made possible thanks to contributions from Matt Caponigro, Diogo Rede, Nishi Grover Garg, Simon Knight, Shanu Mandot, Hannah Son, Heather Vercillo, and Peter G. Walen.