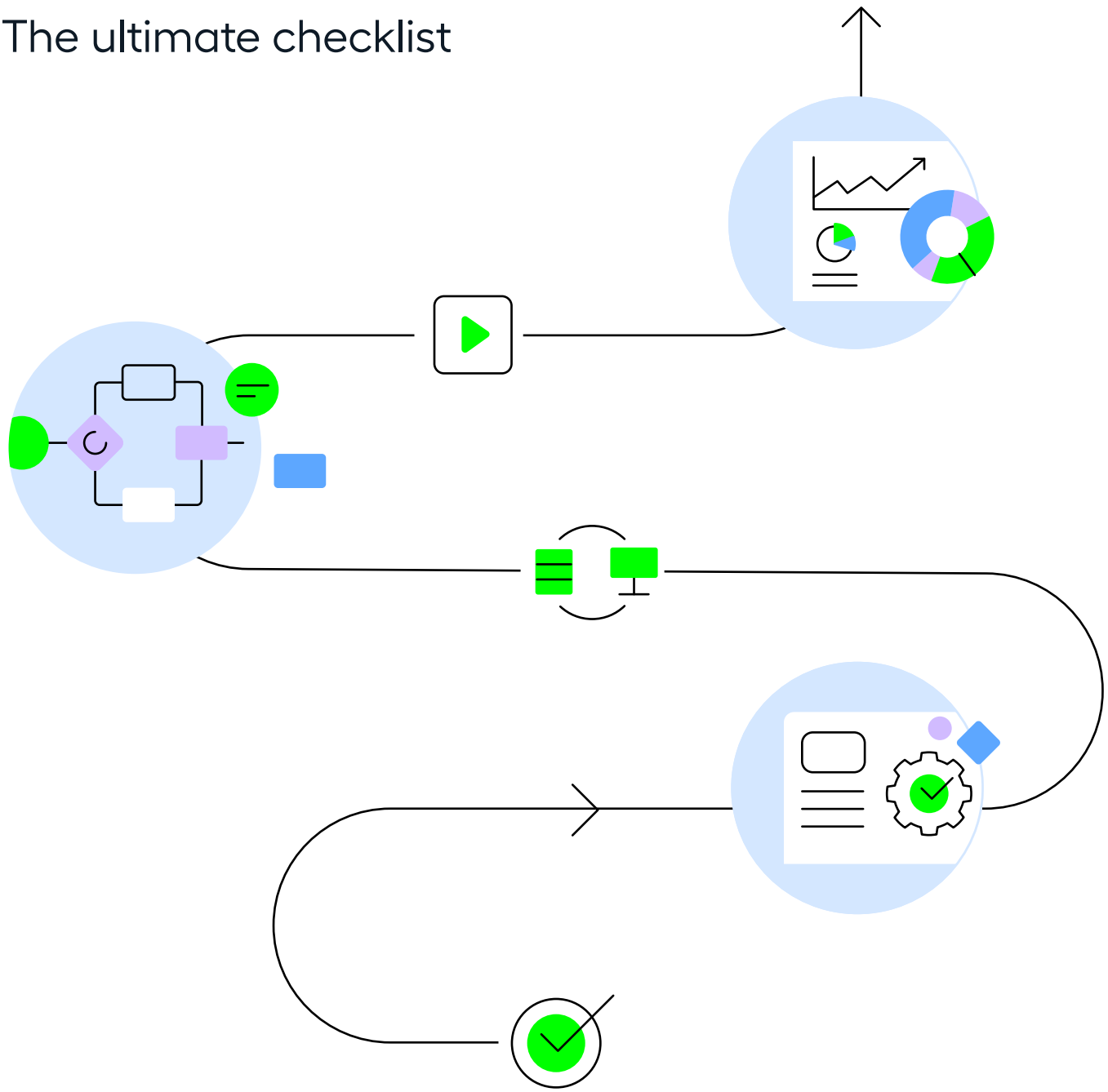
leapwork

Test Automation Strategy in 2025

The ultimate checklist



Covered in this checklist

Introduction

The software development landscape is changing: efficient and effective testing is crucial for ensuring the quality and reliability of applications. Test automation plays a pivotal role in achieving these goals, helping organizations streamline their testing processes, reduce manual efforts, and accelerate the delivery of high-quality software.

This checklist presents a comprehensive approach to creating a robust test automation strategy, outlining key phases in the automation test life cycle and essential considerations for successful implementation. We will delve into critical aspects such as automation approach, risk analysis, environment setup, and execution planning.

By following this checklist, your organization can optimize testing efforts, minimize risks, and enhance test coverage.

A well-defined test automation strategy can be the difference between weeks spent firefighting and a smoothly progressing sprint. Investing the time to develop a robust strategy will save valuable time and resources throughout your automation endeavors.

The test automation life cycle

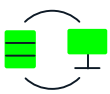
Let's begin by examining the automation test life cycle, which comprises the following phases:

Automation feasibility analysis

In this phase, you assess the feasibility of automation. This includes shortlisting the test cases for automation and setting the requirements for a tool.

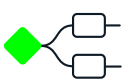
Test strategy

In the test strategy phase, you choose a test automation framework/tool, create a test plan, and develop a test automation suite within your test management tool.



Environment set up

This phase involves configuring the testing environment and acquiring the necessary hardware and software for automated test execution.



Test case development

Here, you create automation test scripts/flows, ensuring they are reusable, well-structured, and well-documented.

Test case execution

In this phase, you execute your test scripts/flows and monitor their performance.

Test result generation and analysis

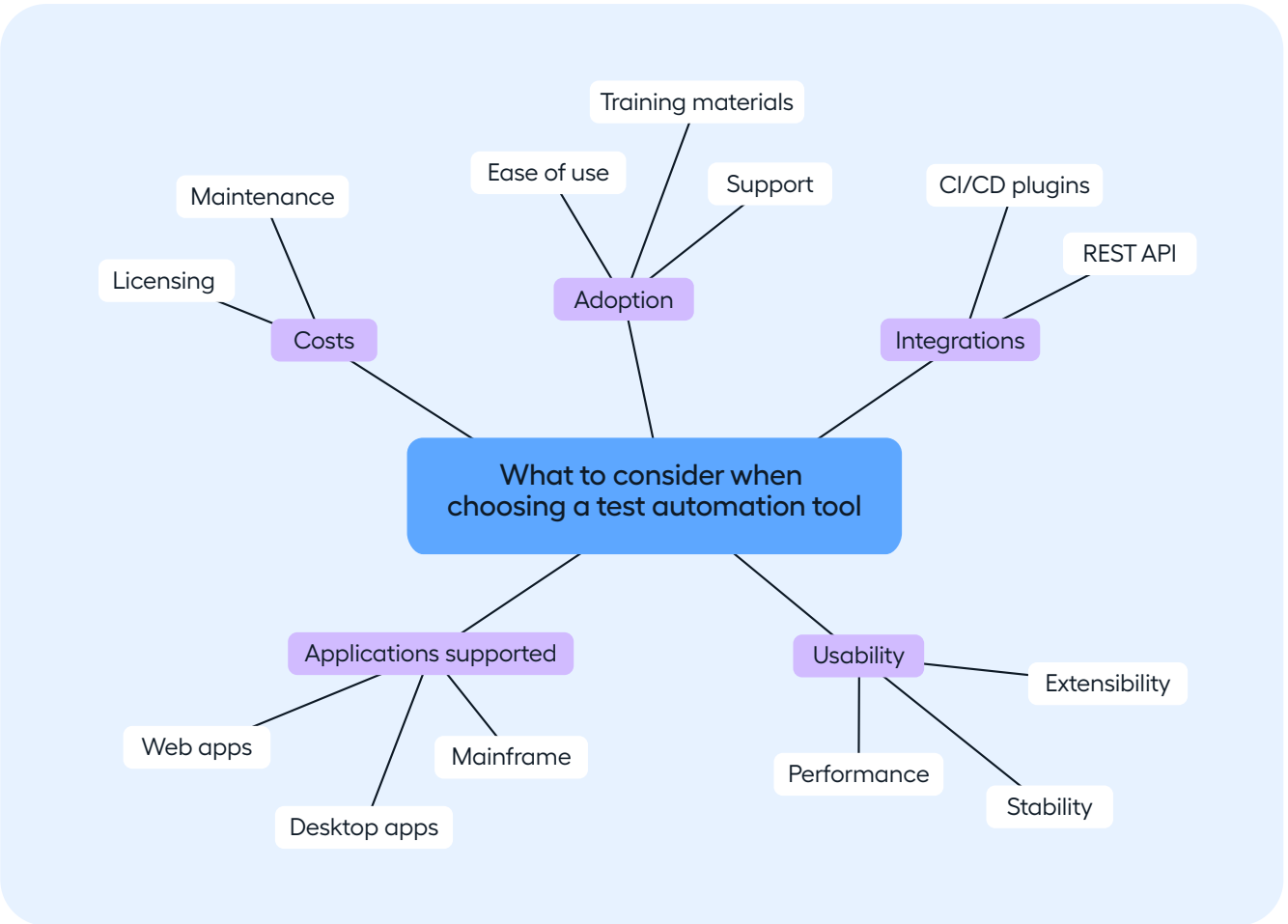
The final phase involves analyzing test case output and sharing reports with relevant stakeholders to demonstrate value and discover areas for optimization.

1. Testing tools

Selecting the right tool defines the success of your test automation strategy and will make or break your test automation project. For this reason, it's the first step in this checklist.

Every organization is unique, with varying people, processes, and technologies. Each QA team faces different challenges while working with various applications in diverse environments. Therefore, aligning tool requirements with these setups is crucial.

Consider factors such as costs, application support, and usability when determining the right tool. Shortlist tools that meet your specific requirements.



Once you have identified your testing requirements, you can shortlist the tools that provide your required features.

Many cost-effective tools are available for automating different applications, including Windows applications, web applications, websites, mobile web applications, and native mobile apps. These tools can be either paid or open-source.

Selenium

A popular starting point for many is Selenium, a free, open-source solution. However, it demands significant time, effort, and resources to see value. The steep learning curve, missing tech support, and lack of reporting opportunities make it a time-consuming and expensive solution long term.

Custom-built automation frameworks

An automation framework is essentially a recipe for how to build an automated test case with code. The benefit of custom-built frameworks is that they can be tailored to fit the precise needs of the organization.

The downside is that their long-term success is largely dependent on the person or people who built them. Coded frameworks require maintenance when the UI of the application they are built for changes. In some cases, every test must be rewritten. With a small number of tests, it's doable. But when your tests run into the hundreds or even thousands, the maintenance burden completely outweighs the benefit of automation.

No-code automation

Many businesses struggle with complex code-based automation tools, limiting their accessibility and leading to resource-intensive maintenance. This often hinders automation initiatives.

No-code automation addresses these challenges, offering user-friendly tools that enable non-technical users to create and execute tests and workflows quickly. This approach enhances productivity, promotes cross-functional collaboration, and reduces maintenance efforts, ultimately facilitating scalability within organizations.

2. Scope

Defining a project's scope from an automation perspective involves setting timelines and milestones for each project sprint. In this stage, you clearly define which tests to automate and which to keep doing manually. Not all test cases can or should be automated.

Use the following list to identify ideal tests for automation:

- [ ] **Repetitive and regression tests**

  Automate test cases that require frequent execution, particularly for regression testing, to ensure that new code changes do not introduce bugs into previously functioning features.

- [ ] **Stable and well-defined requirements**

  Automate test cases based on stable and clearly defined requirements to ensure consistent results and facilitate script creation and maintenance.

- [ ] **High-volume and data-driven tests**

  Automate tests involving extensive data processing or multiple iterations with different datasets.

- [ ] **Critical business logic and functionality**

  Automate tests for critical business logic and core functionalities to ensure their reliability and correctness across different releases and updates.

- [ ] **User Interface (UI) testing**

  Automate UI test cases for critical user flows, interactions, and validations to ensure the user interface functions as expected.

Cross-browser and cross-platform testing

Automate tests that need to be performed across multiple browsers, operating systems, or devices to ensure compatibility and consistent behavior.

Workflow and business process testing

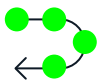
Automate tests that replicate complex business workflows and processes to validate their accuracy and efficiency.

A rule of thumb is the 80/20 split

Select 80% of the test cases that, if automated, would reduce the risk of errors happening during regression testing to an acceptable level. The remaining 20% can then be left for manual testing or not considered part of the current regression suite.

3. Test automation approach

When choosing a test automation approach, consider three areas: **processes**, **technology**, and **roles**.



Process

A test automation roll-out must be a well-defined and structured process. Make sure to cover the following in your plan:

- When during the sprint should automated test cases be developed?
- When are features ready for automated testing?
- How will we take care of maintenance?
- How do we analyze results?

Technology

Identify the applications to be automated, determine their technology, and confirm that your test automation platform supports these technologies.

In most cases, automation will encompass various application types, including web-based, desktop-based, and mobile. Select a tool that can meet all your automation requirements. You should also outline which kinds of tests you're looking to automate. Unit and integration testing are usually an integrated part of development practices and happen earlier than other testing practices, but there is a long list of other test activities that can be automated.

Types of tests that can be automated include:

- **Functional tests:** Testing feature by feature through the UI
- **Performance testing:** Testing the application under load
- **Regression testing:** Making sure all existing features continue tofunction when new features are introduced
- **Cross-browser testing:** Validating the same web-based application across different browsers

Types of tests that can be automated

Roles

Define the automation roles within your agile team. Ensure all team members understand their responsibilities for the automation project.

Examples of roles and responsibilities include:

- **Automation Lead:** Responsible for coordinating and managing all automation activities, defining the automation strategy and goals based on project requirements.
- **Automation Engineer/Test Automation Developer:** Responsible for creating and maintaining automated test scripts and frameworks. Note that some no-code tools won't require any coding, and this role can in those cases be held by a business expert rather than a technical expert.
- **Automation Architect:** Designs the overall architecture of the automated testing framework and guides the automation engineers in creating efficient and maintainable automated test scripts.
- **Automation Tester:** Creates and maintains automated test cases, identifies defects, and collaborates with the development team to resolve issues.

4. Objectives

To measure the success of test automation, you should set clear objectives that align with your business goals. These objectives should be ambitious, but realistic. Common objectives in test automation are:

- Reducing execution time
- Increasing throughput
- Improving test coverage
- Reducing bugs that reach production
- Improving customer satisfaction
- Increasing ROI

Here's an example of objection setting in test automation:

Alignment with organizational objectives

Sync with company-wide KPIs or OKRs

Precision in goals

Prioritize objectives:

- Cut testing time
- Increase speed
- Expand coverage
- Enhance quality

Tangible benchmarks

Set clear metrics:

- 30% faster time-to-market
- 40% reduced costs
- 70% quicker regression tests

Learn more about how to measure the success of test automation in our blog post.

5. Risk analysis

Risk analysis is an essential aspect of project planning, particularly within automation. This analysis involves creating a list of identifiable risks with the following details:

- **Description and relevant metadata**
- **Severity:** What will happen if the risk becomes a reality? How hard will it hit the project?
- **Probability:** What is the likelihood of this happening?
- **Mitigation:** What can be done to minimize the risk?
- **Cost estimate:** What is the cost of mitigating the risk, and what is the cost of not doing it?

Note that a risk plan is a dynamic document. Risks should be added and removed from the list as the project evolves.

Examples of identifiable risks

Description	Severity	Probability	Mitigation	Estimate
We don't have enough trained resources to create test automation cases. This will lead to lower test coverage as more manual regression testing must be performed before release. This might delay the release.	●●● High	●●○ Medium	Contact partners or training providers. Alternatively, prioritize self-studies to train the team in automation.	TBD
Test servers will not be able to keep up with the load from the automated regression tests, which will lead to a high number of false failures in the reporting.	●●○ Medium	●●○ Medium	Contact Operations and make sure that the test servers are configured to cope with the expected load.	Approx. $10,000

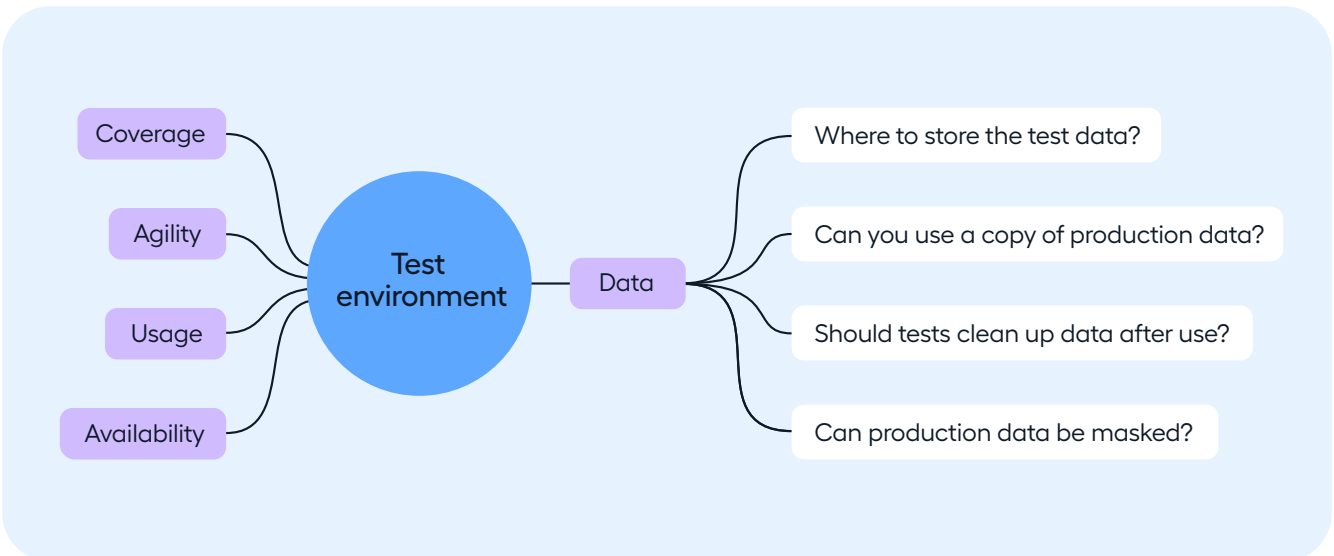
6. Test automation environment

To test software effectively, a stable and predictable test environment is essential for generating reliable automation results. This environment should closely replicate production environments to reflect what end users will experience.

Consider the following aspects related to test environments:

- Where will you store the test data?
- Can a copy of production data be used? This is necessary in some industries
- Can production data be masked?
- Should test cases clean up data after use?

Some release pipelines are already mature and well-defined. Still, it's essential to evaluate the current state of your test environments and consider things suchas environment coverage, usage, agility, and availability.



7. Execution plan

An execution plan should outline the day-to-day tasks and procedures related to automation. Select test cases for automation based on the criteria defined in part 2 of this checklist. Before any automated test cases are added to your regression suite, they should be run and verified multiple times to ensure they run as expected. False failures are time-consuming, so test cases must be robust and reliable.

Follow best practices for setting up your test cases to ensure they are robust and resistant to system changes. Run your tests on a schedule and in parallel to ensure consistent checking of flows at speed.

8. Test naming convention

A consistent test naming convention is a simple yet impactful way to create an effective testing framework.

Here's what a test naming convention should contain:

- Test case no/ID
- Feature/Module
- A brief description of what the test case verifies

Organize test cases logically in a hierarchy or folder structure by name, action performed, or test case ID.

The naming convention needs to be logical and adopted across all test automation contributors in order to have the intended effect.

By following this approach, if a specific test fails during execution, you should quickly be able to understand which functionality is broken by simply checking the test name instead of reading through to understand the complete test case.

Remember, you can never test too much, and the combination of reliable test cases, automated execution, and easy failure analysis will always have a positive effect.

9. Release control

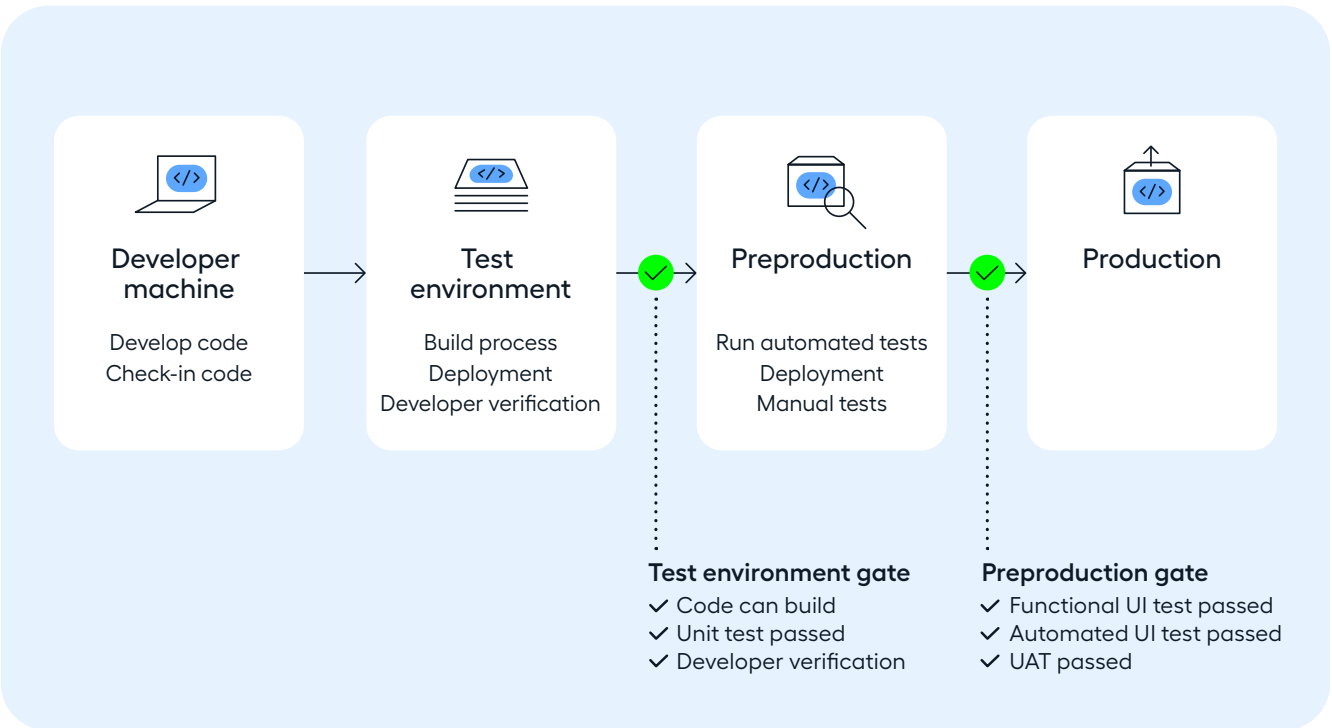
In a release pipeline, regardless of its complexity and maturity, there is a point when a team needs to decide whether to release a build or not.

Parts of this decision-making can be automated, while others will require a human touch, so the final release decision will often be based on a combination of computed results and manual inspection.

In any case, the results from the automated testing should be a key part of the release decision.

Establish clear criteria for passing, such as ensuring that all automated regression tests pass and evaluating application test logs. Whatever the criteria look like in your business, it needs to be clearly defined.

Example of a release pipeline



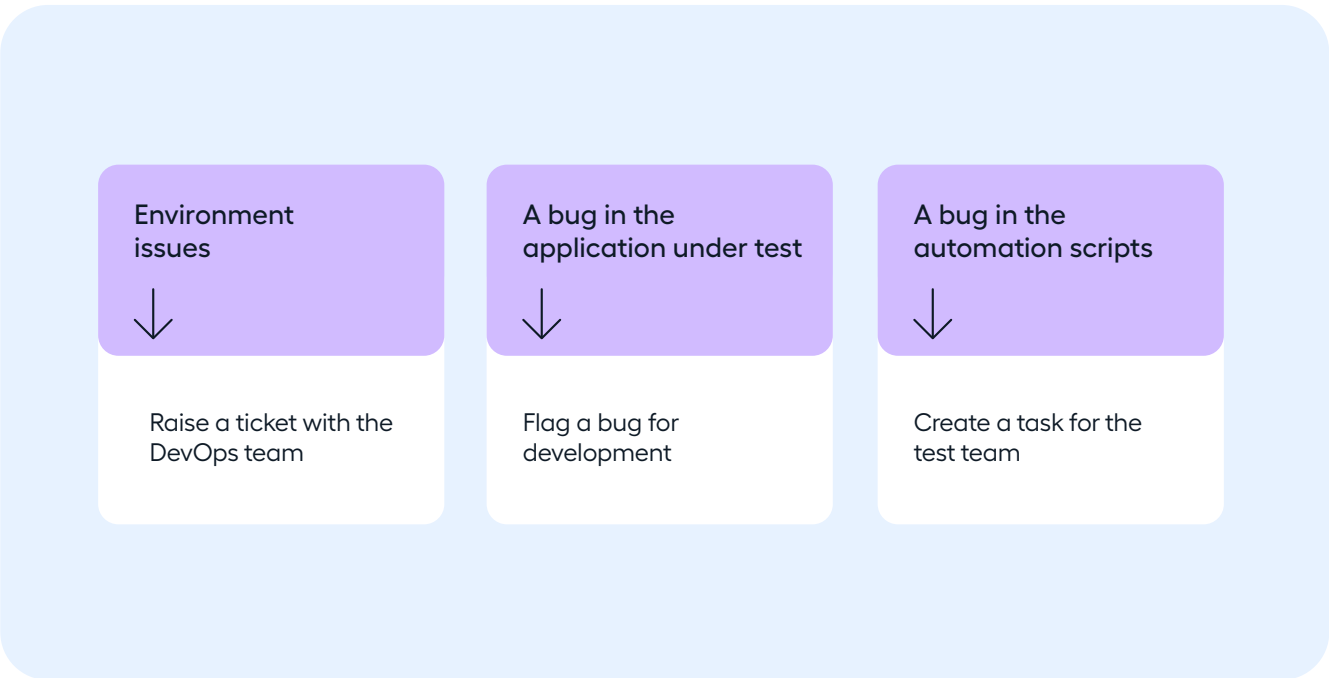
10. Failure analysis

Having a plan for analyzing failing test cases and taking necessary actions is a crucial, yet sometimes overlooked, part of a test automation strategy.

The time it takes from the moment a tester is notified of a failing test case until the fix is described, understood, and accepted in the development backlog, is usually much longer than teams anticipate. Having a well-defined process for this can save a lot of time and frustration for a development team.

Utilize dashboards to quantify and qualify software quality with defined test metrics. This aids stakeholders in understanding the software's status.

Outline how different errors are handled, for example



11. Review and feedback

Finally, once you've made a draft of a test automation strategy, make sure to have it reviewed and approved by all members of the involved development team.

Foster a culture of continuous learning and improvement by embracing feedback from stakeholders, peers, and team members. Lessons learned during software automation should be captured and documented for future reference. Continuously enhance your test automation strategy based on these insights.

Dive deep in our series of test automation strategy webinars

From Manual to Automated Testing: Putting Test Automation on the Agenda →

How to Build a Successful Test Automation Strategy →

From Selection to Success: Finding the Right Test Automation Tool →

Securing Support: Proving the Value of Test Automation →