TEST AUTOMATION **STRATEGY**

A Checklist

Test automation is a cornerstone in DevOps, and when implemented correctly, it helps increase output quality while containing costs. Not surprisingly, IT departments everywhere are realizing the importance of having an actual test automation strategy, instead of just putting out fires here and now.

Once the decision has been made to roll out test automation, the next issue presents itself:

How are we actually going to do this? What's the plan?

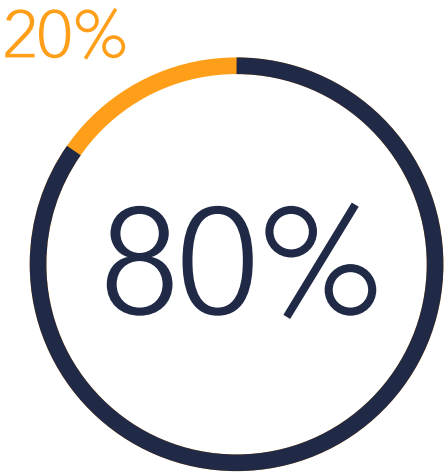
Remember, as with anything in IT, you can 'save' a few days of planning and instead spend weeks testing or programming later—or you can allow yourself to spend a little time developing a test automation strategy to help you save valuable time during sprints.

This is why we've put together this checklist for creating a test automation strategy. It consists of eight items for you to consider as you head out on your automation journey. Some of the items you might already be able to check off, while others will require some work - perhaps even help from external consultants.

1. SCOPE

Defining a project scope from an automation perspective includes outlining timelines and milestones for each sprint in the project. All team members (product owners, developers, testers, etc.) should be on board with the scope.

At this stage, clearly define which tests to automate and which to keep doing manually. A rule of thumb is the 80/20 split: Select 80% of the test cases which, if automated, would reduce the risk of errors happening during regression testing to an acceptable level. The remaining 20% can then be either left for manual testing or not considered part of the current regression suite.



2. TEST AUTOMATION APPROACH

When choosing a test automation approach, there are three areas to consider: *Processes*, *technology*, and *roles*.

PROCESS

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?
- Which features are tested manually?
- Who takes care of maintenance?
- How do we analyze results?

TECHNOLOGY

Identify the applications to be automated. Figure out which technology they are based on and whether your test automation platform supports these technologies. In most cases, rolling out automation will involve several application types: web-based, desktop-based, SAP, mobile apps etc., therefore it's important to have a tool that can handle all your automation requirements.

You should also outline which kind of test automation is needed. Unit and integration testing are usually an integrated part of development practices, but there is a long list of other test activities which can be automated. Define the roles for automation in the agile team. Make sure that all members know who is responsible for which part of the automation project.

ROLES

Learn more about why all members of test team has a role to play in automation. Examples of roles and responsibilities include:

- Automation lead: Responsible for coordinating and managing all activities regarding automation in the project.
- Test case designer / reviewer: Similar to code reviews among software developers, it's important to establish a review process for automated test cases. This means that a tester will typically have at least two roles: test case designer and test case reviewer.

3. RISK ANALYSIS

Risk analysis is of course an essential part of project planning in general, but it is important to consider this specifically in relation to automation as well. The analysis is done by creating a list of all identifiable risks qualified with these details:

- Description and relevant meta data
- **Severity:** *What will happen if the risk becomes reality? How hard will it hit the project?*
- **Probability:** *What is the likeliness that it happens?*
- **Mitigation:** *What can be done to minimize the risk?*
- **Cost estimate:** *What is the cost of mitigating the risk – what is the cost of not doing it?*

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

Note that a risk plan is a dynamic document; risks will be added and removed to the list as the project evolves.

☐ 4. TEST AUTOMATION ENVIRONMENT

Organizations with a software department will have a more or less well-defined method for how software is released to production. This process usually includes one or more test environments. Some release pipelines are mature and well-defined (i.e. a DevOps pipeline) and the work towards fast releases have either already begun or have been deemed not relevant.

In any case, it is important to evaluate the current state of your test environments. Test automation is a "deterministic game"; known inputs will produce predictable outputs. This means that stable and predictable test environments is a prerequisite for successful test automation.

Also consider the data that is part of the tests:

- Where to store test data?
- Will it work to use a copy of production data?
- Can production data be masked?
- Should the test cases clean up data on their own after use?

☐ 5. EXECUTION PLAN

An execution plan should outline the day-to-day tasks and procedures related to automation.

Pick the test cases to be automated based on the approach defined in step #1. Before any automated test cases are added to the regression suite, they should be run and verified multiple times to ensure they run as expected. False failures are time-consuming, so it's essential that test cases are robust and reliable.

Define a set of best practices that make test cases resistant to changes in the system being automated. These guidelines will depend on the application in question, but they should cover how test cases recognize and interact with elements in the application under test.

Execution of test cases should be handled either by the pipeline orchestrator (Jenkins, TFS, Bamboo, TeamCity, etc.) or by a scheduling tool. This means that regression tests will run either as part of a build/deployment event or on a known time during the day. Also consider selecting a setup that allows for parallel execution of the test cases to get the feedback from the regression tests faster back to the development team.

Remember, you can never test too much, and the combination of test automation, reliable test cases, and scheduled/controlled execution will always have a positive effect.

6. RELEASE CONTROL

In any 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, other parts still require a human touch, so the final decision regarding release will often be based on a combination of algorithm results and manual inspection.

In any case, make sure that the results from test automation are part of the release decision. Either decide to only allow releases if all regression test passes or have the lead tester approve the result.

After a complete run of regression tests, consider including the application logs as part of the release decision. If the regression tests have good application coverage, then any errors not related to the UI should reveal themselves in the log files.

Release pipeline

7. FAILURE ANALYSIS

Having a plan for how to analyze failing test cases and the actions required to take afterwards is a critical—and sometimes neglected—part of a test automation strategy. The time it takes from 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 frustrations in a development team.

Outline how different errors are handled, for example

- Environment issues
  → Raise a ticket with the DevOps team
- A bug in the application under test
  → Flag a bug for Development
- A bug in the automation scripts
  → Create a task for the test team
- etc.

LEAPWORK

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.

Enforce a culture for continuous learning and improvement. Include and embrace feedback from stakeholders, peers, and all team members working with automation. And adjust the strategy if needed.

Read the LEAPWORK guide to reducing risk, lowering costs, and driving value with test automation **here**

BOOK **DEMO**

START **TRIAL**