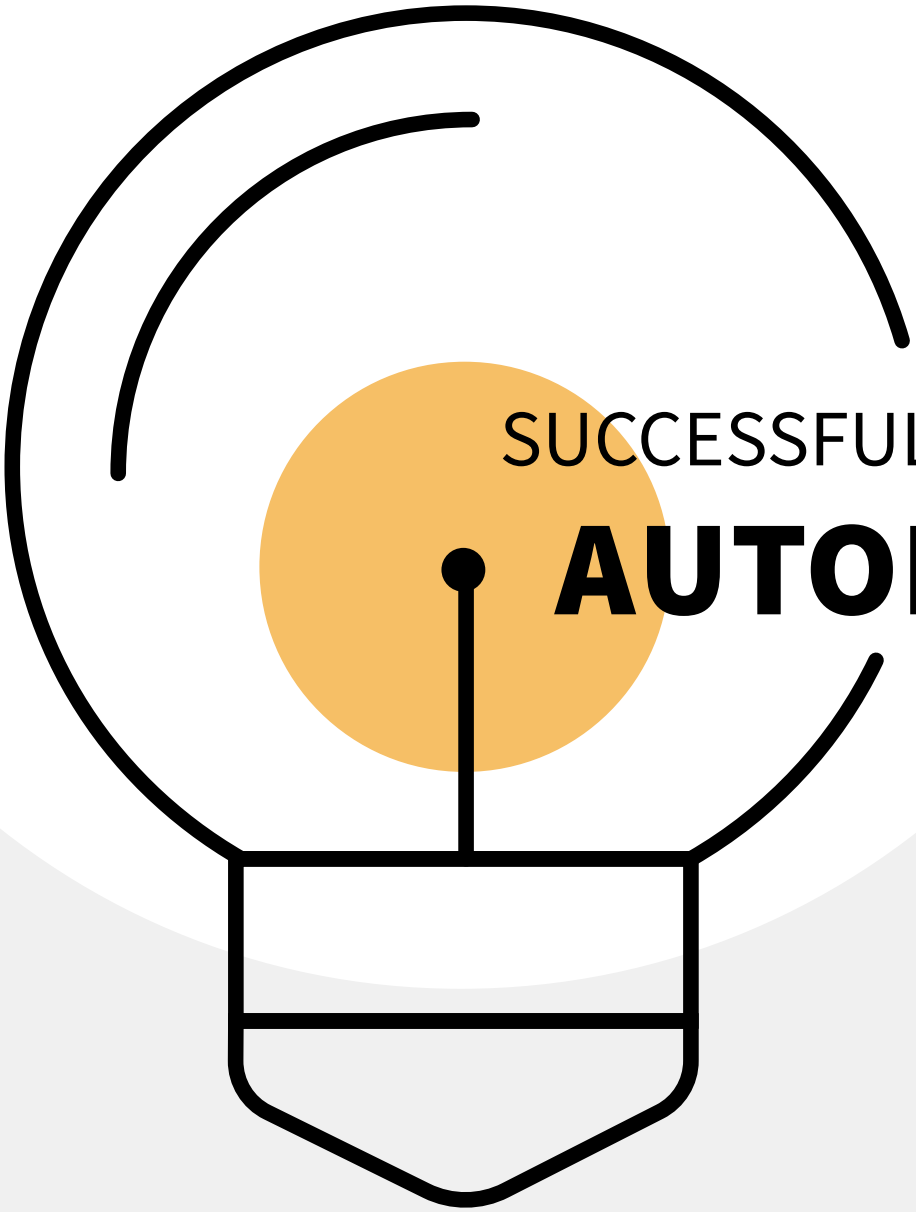
Ranorex®



SUCCESSFUL **TEST AUTOMATION**

10 essential strategies for a successful automation project

10 essential strategies for a successful automation project

Executive Summary

This white paper contains recommendations from test automation experts based on their years of experience in helping customers realize their quality assurance goals. The recommendations are presented in a step-by-step format that can serve as the outline for an automation project. As with any project, the decisions made during the planning stages of a test automation project set the stage for success – or for failure. For this reason, our experts recommend that organizations take the necessary time to set goals, analyze current processes, and build the right implementation team prior to launching an automation project. Our experts also recommend preparing a sample of test cases for automation and conducting an on-site proof of concept to ensure that the selected tools work as needed in your environment. The concluding sections of this white paper suggest strategies for long-term success, to help manage the effort involved in test case maintenance and realize the highest possible return on your automation investment.

Contents

1. Analyze your current processes.

Before implementing test automation, determine why you need test automation and whether your team is ready for it. Perhaps you are looking at automation because the QA process has become a bottleneck in your release cycle. If the manual test cases in your regression set take too long to complete by hand, then there is a good chance that automating these test cases will produce faster results and ease the bottleneck. But, what if the problem is that QA isn't able to start testing until late in the development cycle, due to issues with changing requirements or barriers to communication between the development and QA teams? In such a scenario, your automation project may not deliver the benefits your team expects. Assess whether your organization has the right processes in place to realize the benefits of automation.

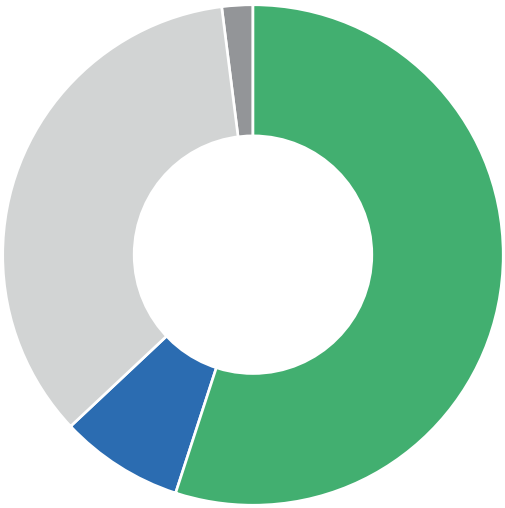
Simply put, test automation won't fix a broken process, especially when the various stakeholder groups have different expectations of their roles in the software development process. Larissa Stoiser, Ranorex senior quality engineer, explains:

> "People sometimes think that test automation will make everything fine. But a team has to have its workflows in place first, and the staff may benefit from training on the purpose of testing and the meaning of quality. They have to live a lifestyle of QA. While important, test automation is just a small part of the larger work of quality software development."

So, it is important to identify any necessary process improvements as early as possible in your automation project. Aspects of your process to consider include the team structure, communication, and development methodology. According to a 2017 survey by Ranorex, while a majority of organizations are using primarily agile development methodologies, a significant number are using mixed/agile waterfall approaches that often have inherent inefficiencies.

Ranorex Survey: Agile vs Waterfall

Development Environment



- Primarily Agile (55%)
- Primarily Waterfall (8%)
- Mixed Agile/Waterfall (35%)
- Other (2%)

For more information on the results and conclusions of this survey, see the Ranorex white paper "Accelerating Toward Continuous Testing: 9 Things to Know About Software Testing on Agile Teams."

Regardless of whether an organization has committed to a fully DevOps approach to development, or is following a more traditional waterfall approach, communication can be improved by creating smaller, cross-functional teams with stakeholders such as developers, QA, sysadmins and business representatives working as closely together as possible. But be sure to create these teams strategically, to allow each team to operate as independently as possible from other teams with a minimum of overlap. Jason Branham, senior sales engineer at Ranorex, observes:

> "I've seen some medium-sized organizations that separate their overall QA team into smaller teams that cover certain views or features of the application without any real strategy. As a result, one feature may have 10 tests while another has 50 tests. So the division of labor is not an automation process, but it has to be done right. If not, automation will be that much harder for you. Communication is key – you don't want people working on the same thing, duplicating their work. The QA process is integral to being successful with automation. You have to know the scope of your tests, why you do your tests – all that has to be documented if you want to have an easy time doing your automation."

In a cross-functional team, stakeholders meet to discuss requirements from the perspectives of business analysts, developers and testers – to ensure that the right product is being built, and that it is being built and tested to deliver maximum value to the user.

Simon Knight, TestRail product manager at Gurock GmbH, describes the importance of communication in ensuring an effective development process.

> “Typically, determining whether the right thing is being built is a requirements question: Do you have a good understanding of what it is that you’re trying to achieve with a specific release or a specific feature? Problems with requirements can occur in any different number of ways. But typically the root cause is one of communication, being able to extract what is in the product owner, manager, or stakeholder’s mind and turn that information into stuff that developers can actually work on.”

In his role as a product manager, Knight often works with teams to help create processes that communicate ideas, concepts, features, and stories more clearly so that developers have a good understanding of what they’re working on, testers have a good idea of what they’re testing, and the team as a whole has good understanding of what risks need to be addressed during the development process and how best to mitigate those risks with clear acceptance criteria and testing.

2. Create an automation project plan.

Depending on the size of your organization, your project plan may simply be an informal review, or it may be a more formal automation feasibility analysis. Regardless of the project size, it's important to identify clear automation goals.

The following questions may be helpful as you think about your goals:

- What is your definition of automated testing?
- What problems are you trying to solve by automating?
- What are the organization's objectives for the automation project?
- How do your test automation objectives support your overall testing goals?
- How does manual testing fit in with your automation testing plan?
- What is your expectation of test coverage with automation?

Create a definition of "success" for your automation project that is realistic and measurable. For example, Ranorex customers typically report reductions of 50% or more in the time required to complete a regression set. So, if your current regression set is taking 16 hours to complete, a realistic success statement could state: the automation project will be successful when the regression set is able to be completed within 8 hours. Be sure that your success statement goes beyond just numbers to identify the true value of the project for your organization. The value statement for the preceding success statement could be "so that our testers are able to conduct time-boxed exploratory testing", or that "so we only expend manual testing resources when the application is stable, defined as passing the smoke and sanity tests."

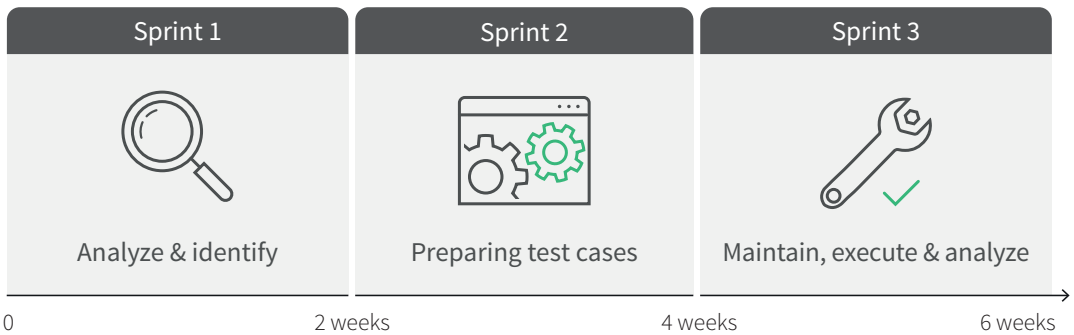
Your project plan should also include your project scope and initial budget. It is also helpful to include information such as:

- What metrics will be used to track the automation project, such as costs, schedule, code coverage, test execution times, number of defects reported?

- What are the project deliverables? These could include the project plan itself, a set of criteria for selecting a tool, results of tool evaluation, results from proof of concept testing, and a body of automated test cases.
- How will the automation project affect other deadlines?
- Where you maintain your centralized repository of test cases?
- Where will the automation team be located? Ideally, the team should be in close proximity to encourage collaboration.
- Where and when will training occur?

When developing your project plan, be sure to allow adequate lead time for acquiring any hardware or software resources that may be needed for the project. Realize that your testers probably won't be able to just squeeze in automation tasks alongside their normal sprint testing activities; instead, treat automation as a development project in its own right. Consider applying an incremental, iterative approach to managing your test automation project, with an emphasis on collaboration. Is your development team using a Scrum process? Organize your automation project that way as well. In your first two-week sprint, for example, you might have stories with deliverables related analyzing your current processes, creating your project plan, and identifying team members. The second two-week sprint might have stories related to preparing manual test cases for automation, researching and selecting tools for further evaluation, and setting up an environment for conducting a proof of concept. An agile approach to test automation has the same benefits as for software development: it helps maximize your chance of success by encouraging collaboration and smaller, more frequent deliverables, and rapid response to changing requirements.

Finally, once test automation is in place, it will need to be maintained. Ensure that you have the long-term resources to maintain and execute your automated test cases, and analyze their results.



3. Put together the right team.

Bringing in the right people to your evaluation and implementation team will be critical to your project's success. All successful automation projects will involve one or more champions. While this person does not have to be a test automation engineer, such champions do often have a relevant background in quality assurance, database, or software development. The key factor is the desire and ability to help the automation project move forward. Other essential team members include representatives of all stakeholder groups, such as one or more developers, testers, business analysts, and sysadmins. This team should meet on a regular basis to set goals, monitor progress, adjust strategy, and sign-off on the final solution.

Jason Branham explains the need for a test automation champion:

> "What we see in practice is that most teams won't attempt test automation without having at least one subject-matter expert on the team, who has some automation experience in their background. But it's not a requirement that they have an automation background because whether you're a manual tester or a full-blown automation engineer, you can find a toolset that can work with your talents."

The test automation champion may be someone internal to your organization, or you may find it beneficial to bring in an outside consultant. For example, Simon Knight worked for many years as an external quality assurance consultant, assisting clients to help identify whether or not the right solution was being built, and whether it was being built right. His consulting work sometimes involved partnering with an existing testing team to implement testing functions, or to put in the right processes in place, or to assist with tools to help automate testing processes.

Likewise, Ranorex used an external consultant to develop its own internal process for quality management. Larissa Stoiser recalls,

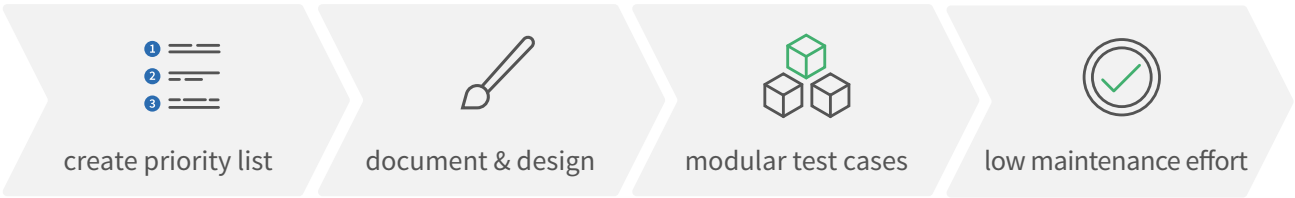
> “As a software company developing a tool for quality, we were naturally aware that quality is very important. But still, like any rapidly-growing company, we had some inefficiences in our processes. So we brought in an external consultant who was very experienced in quality processes. He helped us to create a quality plan for the entire company. For each department, the plan documented their point of view on what product quality means, and what they expected from a quality plan. The company then agreed on quality goals for each department for the next year. After the initial year of the quality plan, we had a follow-up with the consultant to review our processes and make adjustments where necessary. We found that having an outside expert was very helpful in developing an optimal quality process. Now, our current process is quite agile and lean.”

Currently at Ranorex, the development teams have full responsibility for the quality of the products and features they produce. Stoiser explains:

> “We have to think about everything: what the customer wants, schedule our time, and all this freedom and responsibility have the development team very happy. While the team includes staff with a testing background who primarily do testing, the developers themselves are also very involved in the testing process. We try testing as soon as things are there in the code to test. It’s very agile, short feedback loops, that’s the current process.”

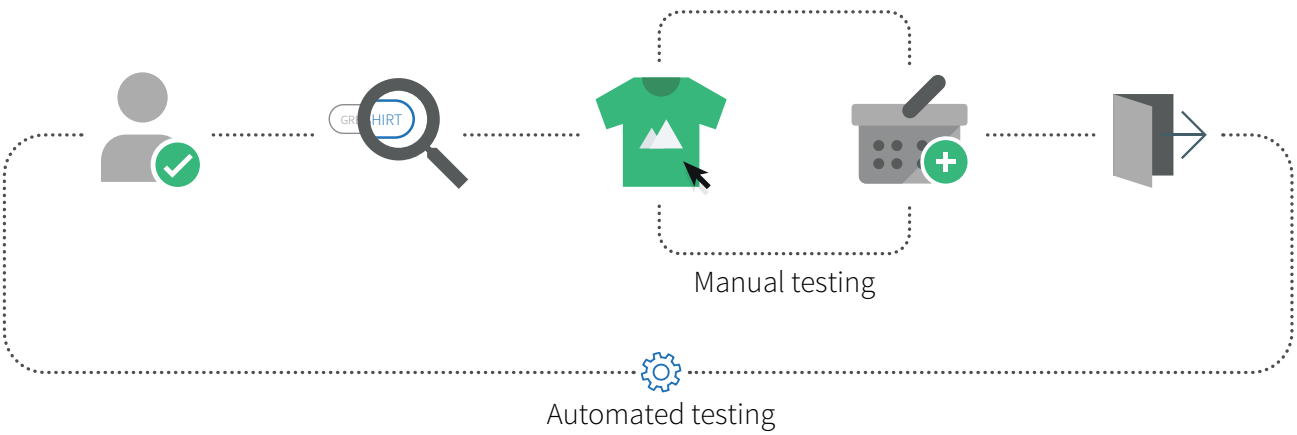
It is easier to implement test automation when developers have a quality mindset and are committed to creating testable code – for example, code that is modular, and that has consistent IDs for UI elements. To help encourage this mindset, allow the team the freedom to help define roles and responsibilities for the automation project and for maintenance of the automated tests after the initial implementation. For example, identify who will be responsible for creating and maintaining tests and object repository information. Who will execute the tests? And who will be the in-house support person for the tool? One of the key benefits of creating a cross-functional implementation team is to generate a shared commitment to quality from all members of the team, developers as well as testers.

4. Prepare a handful of test cases for automation.



Develop a priority list of test cases to automate as part of your tool evaluation. Good candidates for this list include smoke tests, build acceptance tests, and high-risk regression tests. Be sure to include at least one end-to-end test case, to ensure the selected test automation tool works with your key functionality.

After selecting several pilot test cases, ensure that they are well-documented and well-designed. It's common for manual test cases to need additional information to be ready for automation. For example, a web shopping site may have a test case that describes the expected behavior of the application when a user places an item in their online shopping cart. The shopping cart test case may not include preconditions for reaching this point in the application, such as logging on or searching for an item. That may be fine for a manual test case because a tester can interpret the instructions and complete the missing steps. But the steps must be written explicitly for an automated test case. It is important to document any preconditions for a test case that will be automated, along with the step-by-step instructions to execute the case, test data values, validations to be performed, cleanup steps needed after the case completes, and criteria for determining whether the test case was successful.



If you do not currently have a method for documenting your test cases, be sure to set up a template. The template could be a spreadsheet file or a Google form, or you could use a test management tool like TestRail that can maintain a repository of test cases with preconditions, test data, expected results, and estimated effort.

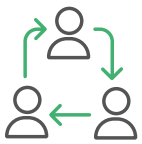
In addition to preconditions and test data, automated test cases will need rules for error handling, such as how to respond to unexpected events such as pop-up windows. This is a separate step from the defect detection function. To save on maintenance, be sure to make your automated test cases modular – which is often not a key consideration in manual test case planning. For example, while a manual test for a shopping cart may not include steps to start the application and log in, an automated test case will need this functionality. But instead of including the login procedure in each automated case, your team should develop a single login test case, and then invoke it from other test cases that have this as a prerequisite. This type of modular design will reduce your test case maintenance effort.

5. Research and select two or three top automation tools for further evaluation.

When considering automation tools, the primary criteria is the tool's ability to automate tests for your particular desktop, web or mobile technology. For example, if you are testing a web-based application, then the ability to distribute cross-browser tests on a grid may be important for you. For mobile applications, a solution that can automate tests for Android and iOS applications on both real devices and on simulators or emulators will be important. Additional evaluation criteria that are independent of your development technology are listed below:

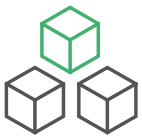
Use of a standard, full programming language vs. a scripting language

IDE with support for debugging and refactoring



Tools to facilitate team collaboration, including reusable objects

Support for codeless automation for use by non-programmers



Framework support – modular, data-driven, BDD, keyword-driven, hybrid

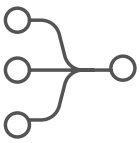
Automation able to be independent of the AUT, without jailbreaking

Strong UI object identification

Handle operations through GUI and/or backend processes

Flexible licensing options; such as multi-user licenses or run-time licenses

Integrations to other tools



Support for version control

User-friendly interface

If your team includes experienced automation engineers, you can naturally draw on their prior experience with automation projects or that of other colleagues in the field. It may also be helpful to read independent reviews of products, either in trade publications, research by analysts such as Gartner or Forrester, or online at sites such as G2Crowd or Capterra.

6. Do a proof of concept.

Once you have selected two or three tools for further evaluation, conduct a proof of concept (PoC). This is a short-term project, typically lasting a few weeks, in which your implementation team will evaluate the leading tools in your environment, and measure the tools against your evaluation criteria. During the PoC, you will attempt to use each tool to automate the handful of test cases described in recommendation 4, above. If you are considering one or more paid tools, contact the vendor's sales department for assistance in conducting the proof of concept. The vendor's pre-sales engineer will typically assist you with obtaining one or more free evaluation licenses, installing the software on your hardware, and overcoming any technical obstacles to automating tests for your application.

Jason Branham describes the Ranorex approach to assisting in a PoC:

> "We're more than just product support. We want to make sure that Ranorex is a good fit for the customer's infrastructure, processes, and team. So, we do a multi-faced analysis. We want to check all the boxes – not just to prove that it works, but that Ranorex really is the best fit for their needs. So we may do some training through demonstrations, provide support on the product where something isn't working due to issues in their environment, whatever it takes."

Following the proof of concept, your team may be prepared to recommend a tool for adoption or may decide that alternate tools should be considered.

7. Implement the selected automation tool.

Once you have selected an automation tool, the process of implementing your automated tests can begin. Set up the test environment and configure any necessary integrations, such as defect tracking applications, test management solutions, CI/CD servers, and so forth. If you have chosen a paid solution, ensure that you have the necessary licenses to develop tests on local machines and execute them on remote servers.

Because test automation is a software development project in its own right, consider using a source control system such as Git for your test cases. Larissa Stoiser shares Ranorex's internal approach:

> "Our Continuous Integration development process includes Team Foundation Server (TFS) from Microsoft for work item tracking. Our version control is maintained via Git in the TFS. Our testing code which is generated from Ranorex also is stored there. With Git, you have the freedom to easily have each bug fix on a separate branch, and therefore you are also agile in your testing. One approach is to maintain the testing source and the Ranorex source code in the same repository to increase efficiency and have the right tests in place for the right branch."

In addition, ensure that your test environment itself is sufficiently stable to support test automation. Simon Knight describes what can happen if it isn't.

> "I worked with a client who was basically building an application on top of some other third-party analytics platforms. They had various flavors of off-the-shelf analytic servers, plus the application that they were building, and various databases dependent on the application, and other sorts of subsystems as well. Now, the other systems were peripheral: We weren't testing the databases and analytics platforms, just the application they were building. But the environment against which they were trying to implement automated testing for their application never performed

> sufficiently to be able to develop automation that was reliable. Things would time out, for example. There were multiple problems that weren't related to the application itself but entirely related to the environment where all of these systems were being hosted. It is a complete nightmare when that happens. You spend all your time trying to figure out if there is a problem with the automation or the environment, or if there is an actual defect in the application, and it becomes a huge time sink. ”

To avoid this, isolate the components of the application under test (AUT) and ensure that your infrastructure is sufficiently stable and reliable for that testing. Otherwise, your team may face significant challenges in test automation.

8. Allow time for training and the learning curve.

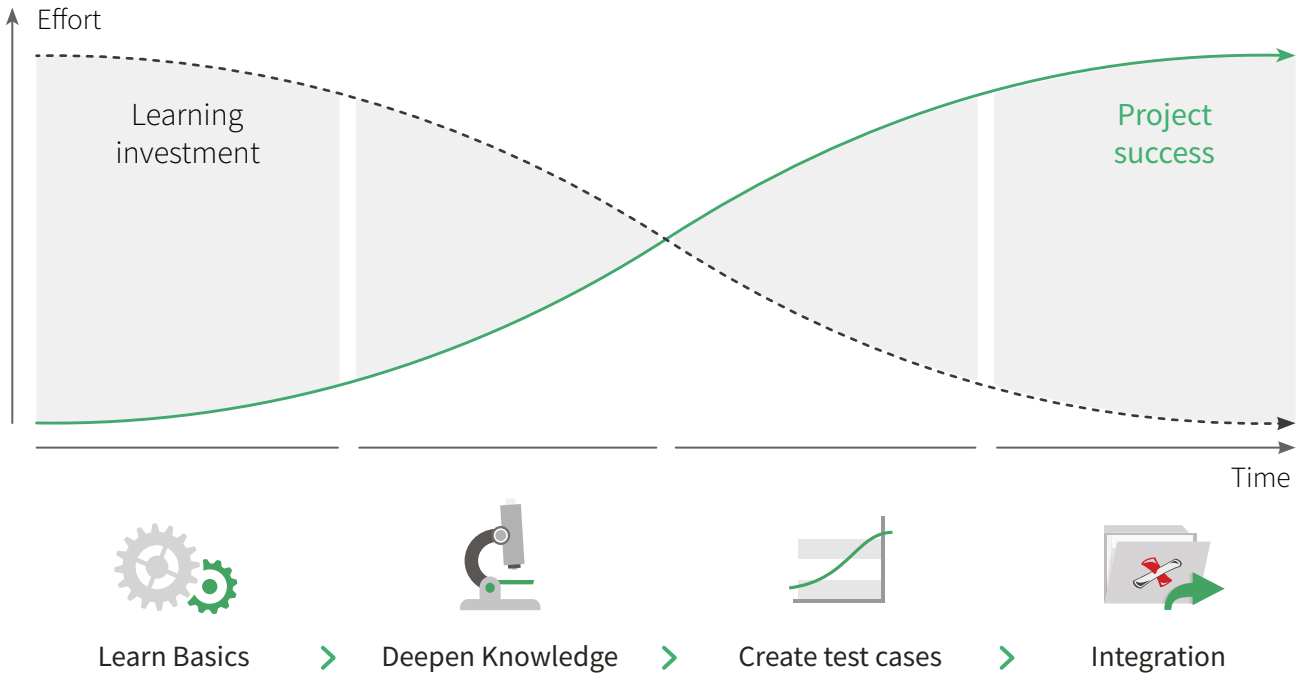
When facing time pressure, it can be tempting to reduce or underestimate the time needed for training on the new tool. But unless your automation staff is already familiar with the selected tool, it is important to schedule adequate time for training. Just-in-time testing will be the most effective approach because your staff will understand the relevance of the training for their jobs and therefore be more motivated. So, the ideal time to schedule automation training is immediately before testers are expected to begin producing automated test cases. Your training plan may include vendor-provided classroom sessions, online tutorials, or a combination of both.

Hubert Gasparitz, senior sales engineer at Ranorex, provides his experience on test automation training:

> “Whether or not training is necessary depends on the team. When you have a new team that is unfamiliar with automated testing, or you have a new tool, you can avoid major issues and get your project running easier and faster with training. Even if your team has experience with another automation tool, training can be beneficial. What I’ve seen is that sometimes automation engineers have experience with a different tool, so they expect the new one to work in the same way as the old one. That knowledge could be a disadvantage because each automation tool is unique.”

You will also need to plan for a learning curve at the start of the project. Even with an easy-to-use tool like Ranorex Studio, your testers may be less productive at the start of an automation project as they come up to speed, as shown in the following diagram. The time investment will pay off in the long run.

The Ranorex evaluation process goes through four phases:



9. Begin automating your documented test cases.

With your test automation tool set up in your environment, and the automation team trained on its use, you are now at the phase in your project where you can begin automating and executing your documented test cases. Expect that this initial cycle of creating and executing automated tests will take more time than executing manual tests, but with each subsequent release cycle, the time to automate and execute your tests will be reduced. Following are additional recommendations for successful test case automation.

Automate where there is a high probability of success.

During the initial phase of implementation, focus on creating high-value automated tests that are valid ("green") and stable, so that your testing staff has a sense of achievement and gains confidence in the automation tool. High-value tests include those that are painful, repetitive and/or failure-prone when done manually, and that are relatively easy to automate, such as those with no dependencies to other systems, external services or databases.

Automate where there is maximum potential ROI.

Next on the priority list of automation include your smoke tests, build acceptance tests, and high-risk regression tests. A quick way to calculate the automation benefit is to determine which tests will give you the highest ROI to automate, and what failures may cause the most damage if a defect makes it into production. For more information on prioritizing test cases according to risk, refer to the section on prioritizing test cases for risk-based testing in the Ranorex Guide to User Interface Testing. Other good candidates for automation are tests that require data entry, cross-platform tests, tests with multiple field validations, and CRUD (create, retrieve, update, delete) actions. Automate repetitive, tedious scenarios to free testers to do manual testing of more challenging and difficult-to-automate scenarios. Look for opportunities to make existing manual tests smarter, such as making them data-driven.

Automate for maintainability.

As you automate, be sure to keep your tests independent of each other with reusable units, modules or methods. Don't copy and paste test steps between test cases. Be sure that your automation includes a step to clean up your AUT after each test.

Be realistic.

Realize that you can't automate everything. Some tests will require so much effort to automate that it would be more feasible to conduct them manually. Jason Branham provides one example: "We recently had a Point of Sale client that was testing the functionality of their check-out process. They had a manual test that included swiping a payment card. In their case, it was simpler to physically swipe a card than to try to automate that functionality. Parts of your testing scenarios where a user would have to do something manually, such as respond to a Captcha, prove they're not a robot—those are put in place to block automation. You either have to get creative or just decide that this test case is out of scope." Part of this is to ensure that you are selecting the right level of automation for each case. Don't GUI test an installation, for example, if the same test would be more efficiently done through unit or integration testing.

Save time by using capture-and-replay features to create your automated tests quickly.

Jason Branham explains, "Some Ranorex customers code everything by hand in the IDE. But the record-and-playback feature of Ranorex Studio actually works really well. You could have your manual test case on your desk, record you test as you normally would, and by the end of it, you have an automation done by Ranorex. It may work perfectly, or you may need to enhance the automation with user code. But this gives you a faster start than coding everything from scratch."

Encourage developers to create application code that is testable.

For UI testing, this means setting automation IDs. The more structured and modular your software architecture, the easier your automation team can write tests for it.

Handle test failures.

Just because one test fails in a test run, doesn't mean that you want all of the others to fail as well. If your selected tool has them, take advantage of customizable settings to control the behavior of a test run when one test case fails, including options to continue with the next iteration of the test case, jump to the next sibling test case, or stop the run with a failure message. When failures do occur, don't immediately assume that the problem is with the application. Rather, analyze each failure to determine whether there is an issue with the automated test case, the environment, or the AUT.

Plan for test case maintenance.

As the AUT changes, incorporate these into your test plan and automate these new test cases during a maintenance window, such as in the early stage of the following sprint.

10. Periodically review your process and adjust as necessary.

After the initial phase of implementation is complete, your project team will review the project metrics and analyze the team's performance. As part of the project review, assess the effectiveness of your organization and communications. Consider how your test automation project has affected the role of testers on the team, and how continued automation may drive additional change.

One of the benefits of a cross-functional team is that it allows your testers to grow in their understanding of software development, even if they do not become developers. Larissa Stoiser describes the changes that cross-functional teams have brought to Ranorex:

> "Our main goal has been to bring QA and Dev closer together and to involve QA from the very beginning in the development process. Even if they do not become developers, we encourage our QA staff to at least write test code, do unit testing, and understand the developers better."

Below are the results of a survey that Ranorex conducted in 2017 on the changing roles of software testers in agile teams. The first chart shows that about 37% of testers on agile teams help write user requirements and define acceptance criteria, responsibilities which typically fall to business analysts in waterfall environments.

The second chart shows development responsibilities of testers on agile teams, with over 50% responsible for creating automated test cases and 22% working with developers to conduct unit and interface/API testing. These survey results show how an automation project in combination with cross-functional teams can help testers develop from an "I-shape" team member with deep testing skills, into an even more valuable "T-shape" team member with both deep skills in testing and broad skills in all aspects of development.

Ranorex Survey: QA/Tester Planning Responsibilites

What planning tasks do testers on agile teams do regularly?

- **78%** Create test artifacts such as scenarios, cases and scripts
- **67%** Participate in creation of test plans
- **65%** Identify functional or non-functional areas to test
- **51%** Set up the test automation framework
- **36%** Help write testable user requirements
- **37%** Define acceptance criteria
- **22%** Participate in risk analysis

Ranorex Survey: QA/Tester Project Responsibilites

What project tasks do testers on agile teams do regularly?

- **54%** Create automated GUI test cases
- **53%** Analyze completed tests
- **52%** Conduct exploratory GUI testing
- **50%** Document/archive tests for regression testing
- **46%** Conduct scripted GUI testing
- **34%** Create post-testing artifacts such as coverage
- **22%** Work with developers to conduct unit, interface/API testing
- **11%** Coach developers in testing techniques

After the initial implementation phase, it will also be beneficial to review the quality of your automation. For example, Ranorex offers a project review service for Enterprise customers for the first three months of their contract.

Bernhard Seunig-Karner, Director of Customer Support at Ranorex, explains how the review works:

> “The customer sends us their Ranorex solution, and gives us input about their desired solution, which includes describing the AUT, and what the test approach is, what they want to get out of Ranorex, and what benefit it should bring them. We look through their automation project for inefficiencies. This is both a human review and an automated one. The automated review creates a list of potential issues, and then our engineers analyze the list to verify whether the issue is genuine or not. After our review, we are able to provide the customer with feedback on what could be improved to be faster, more stable, provide better test results, and be more maintainable.”

Seunig-Karner describes some causes of project inefficiencies.

> “Most of the time, the issues are with the structure of the project. It’s not an issue unique to Ranorex. Rather, it’s an issue of making the structure easy to maintain by anyone. For example, having several test modules that do the same task creates a maintenance issue. Ranorex offers reusable modules to minimize maintenance, but you have to take advantage of this feature. Another issue that we sometimes see is with the structure of the object repository. It’s important to maintain the object identification properly to eliminate redundancy. To avoid this, it’s important to check your recording modules and the object repository to ensure that UI elements well-structured and correct, and there is no redundancy. Likewise, you can have multiple steps in a recording module for what is really a single action. Just like with your application code, sometimes it’s necessary to refactor your automated tests for maximum efficiency.”

Seunig-Karner’s final recommendation has to do with the amount of detail that your automated test reports generate. “If you have a test that runs overnight, it might not be necessary to log every single action into the report. Because then the report becomes huge, and it could be difficult to open because there is so much data – let alone try to analyze it. There are settings in Ranorex that allow you to select just the failures in the test run report. If a module succeeds, then it doesn’t create an entry in the report. This is just one example of the customization settings that can make a test run report more useful.”

Conclusion

To maximize the potential for success, we recommend that you view your test automation project as a distinct effort, separate from development projects. Follow an incremental, iterative approach to managing your test automation project, with an emphasis on cross-functional team collaboration. Do a proof of concept to ensure that the selected automation tool will work in your environment, on your testing platform. Focus on automating those test cases that will be the easiest to automate, yield stable tests, and give you the maximum ROI. Following your initial implementation cycle, review your project and process to identify additional opportunities for improvement. And a final tip: to maximize the return on your automation investment, consider leveraging your automation tool for other tasks, such as loading test data for manual tests or monitoring your production environment.

Ready to start your test automation project? Ranorex can help.

Ranorex test automation engineers are ready to assist you in making your automation project a success. Reach out to our sales team to learn more. Or, if you are ready to conduct your own proof of concept in-house, download a free trial version of Ranorex today to experience the power of test automation.

Contributors:

Jason Branham, Sales Engineering Manager, Ranorex Inc. (USA)

Jason has been with Ranorex since 2014, but has been in automation for over 15 years with experience both in software automation and hardware automation such as relays, switches, robots. He enjoys working with customers to be successful and realize the dream of automation.

Hubert Gasparitz, Senior Sales Engineer/Team Lead, Ranorex GmbH (Austria)

As a pre-sales engineer at Ranorex, Hubert assists organizations with automation tool selection, test automation projects, setting up their test environment, maximizing automation efficiency. He has a background in electrical engineering, software development, and economics.

Simon Knight, TestRail Product Manager, Gurock Software GmbH

Simon has spent the last 10 years working as a tester, test lead, test manager, test automation developer, and testing consultant. Currently, he is the TestRail product manager at Gurock, responsible for defining the TestRail product roadmap and working with the development team to ensure the delivery of a quality product.

Larissa Stoiser, Senior Quality Engineer and QA Team Lead at Ranorex, GmbH (Austria)

A graduate of the Technical University of Graz with an MS degree in Computational Mathematics, Larissa leads the Ranorex quality assurance team. She is an ISTQB Certified Tester and is also a professional photographer.

Bernhard Seunig-Karner, Ranorex, Director of Customer Support

Bernhard has a background as a software developer in both C# and PHP, which he used for firmware development as well as for his bachelor's degree project. He worked in software integration for several companies and hospitals prior to joining Ranorex. Currently, he leads the customer support team, where he ensures that customers get high-quality support, and provides guidance on customer needs to the Ranorex development team.