Test automation is a booming area, however, it is still a very immature area. Many of the successes in test automation projects are due to empirical trial and error processes.

To make matters worse, test automation is still the subject of myths that generate erroneous perceptions about its real benefits and limitations. Often, testing automation tools are overvalued generating false expectations, the required infrastructure is underestimated, among other common problems that will be presented throughout this article.

Best Practices in Test Automation:

In the following sections will be presented some good practices and advice gathered by the author during the last years (at the expense of empirical **learning in test automation projects**).

Test automation is not a testing process

Many companies fail to implement test automation due to the reversal of priorities caused by the pursuit of quality at any price. In this scenario, companies acquire **a test automation tool prematurely** without at least having a minimum degree of maturity in the testing process. The testing process is informal, responsibilities are not well defined and professionals do not have the appropriate profile or are not qualified.

According to Watts S. Humphrey, creator of the CAPAbility Maturity Model (CMM), the quality of the product is directly proportional to the quality of the process used in its life cycle. The deployment of test automation depends on mature and consistent manual testing. Successful test automation projects are those that are based on formal and structured testing processes. After all, how can you expect **something from automated** tests that are based on wrong, ambiguous, or inconsistent manual tests. It is not possible to automate chaos.

Strictly speaking, successful testing automation deployment depends on understanding that test automation or an automation tool alone will not improve or organize existing tests. It is necessary to first "clean up" and implement a formal testing process. The need to automate testing will come naturally as a result of the evolution of the maturity of the testing process.

Automate critical testing first

Nine out of ten test automation projects make the mistake of turning all manual test cases into automated test scripts. The reader must be wondering: But wouldn't this be the primary goal of test automation? Yes and No.

Test automation aims to reduce human involvement in repetitive manual activities. However, this does not mean that test automation should be limited to doing repetitive and boring work. Manual test cases were created in a context where the tests would run manually. It is very likely that these manual test cases are not very efficient in a context where the tests will run automatically.

Often, before starting automation, tests should be redesigned to increase the likelihood of revealing a defect that has not yet been found. After all, the great **benefit of test automation is** not the execution of the tests faster and at any time of the day or night, but the increased amplitude and depth of coverage of the tests.

In practice, automating 100% of manual tests is not always the best strategy. Test automation is ineffective when tests are complex and require Inter system interactions or subjective validations (these types of tests must remain manual). In addition, the cost and time to automate a project's tests are often greater than the cost and time of the development project itself (which would make it impossible to automate 100% manual testing).

The choice of the candidate automated tests, that is, the most critical, should be carried out based on the context of the automation project. However, although there is no widespread categorization, experience has shown that candidate tests are typically grouped into four distinct areas:

- **Smoke Tests**: A minimum set of tests is selected to validate a Build or release before the start of a test cycle;

- **Regression Tests**: Tests are selected for the purpose of retesting a feature or the entire application;
- **Critical Features**: Tests are selected in order to validate critical functionalities that can bring risks to the business;
- **Repetitive Tasks**: Tests are selected in order to reduce the involvement of testers in repetitive and error-prone manual activities, such as mathematical calculations, simulations, processing, file or data comparations , etc.

Incorporate stability into the application

As a rule, tests are automated using the application the way it is. That is, the tests are created from the point of view of the graphical interface and with the sole purpose of automating the actions of end users.

However, testing through the graphical interface is not always the best option for performance-demanding tests. In addition, sometimes the graphical interface does not provide evidence that the test was performed successfully, after all, not always the message "This operation was performed successfully" means that the operation was performed successfully.

Experience has shown that incorporating stability into the application is a key factor for **the success of** a **test automation project**. Stability is an attribute that determines the ability of an application to be tested, that is, the ease of testing an application is directly linked to the level of stability built into this application.

It is usually necessary to make modifications to the application to make it easier to test. These modifications are intended to incorporate a set of mechanisms that facilitate the observation and control of the state of the internal components of the application. Additionally, these mechanisms expose to the outside world the functionalities of the application through APIs, Command Line Interfaces, Hooks etc.

This, in turn, makes the application much easier to test from the point of view of test automation. The purpose of these mechanisms (APIs, Hooks) is to provide interfaces to the outside world that is not dependent on the graphical interface of the application. In this way, you can create specialized tests to exercise some features of the application without the need to use a graphical interface.

Test automation tools also have defects

Test automation is ultimately running software to test other software. Thus, we can affirm that test automation tools suffer from the same types of problems as conventional applications, i.e.: defects. Delays in automation projects are very common due to problems caused by the test automation tool. Among the most common problems, we should highlight:

- Incomplete and ambiguous manuals;
- Non-reproducible defects that occur randomly;
- Memory leak when the tool runs for many hours;
- Performance degradation when the tool runs for many hours;
- Crashes during the recording or execution of tests;
- Incorrect reporting of tests that run successfully when problems occur or vice versa;
- Features that don't work the way they should;
- Incorrect recognition of components (buttons, fields, menus, etc.) of the application under test;
- Regression defects (features that no longer work correctly after a tool upgrade);
- Critical defects that prevent the use of the main features of the tool.

Based on what has been exposed, experience has shown that the following actions must be taken before purchase and while using a test automation tool:

- Create a proof of concept before purchasing the tool to make sure it does not have critical defects;
- Always use the most current version of the tool;
- Do not use the most current version of the tool (upgrade) without performing a regression test (to prevent the fix of a problem from creating worse problems);
- If there is no solution to the tool's problems or limitations, consider creating workarounds using the tool itself.

Demo is not proof of concept

The wonders offered by automated testing tools are typically presented by demos (Demos). Typically, these tools are purchased based on the good impression caused during these presentations.

Sad mistake, unfortunately. Demos typically present very simple usage scenarios in a controlled environment. Sometimes that perfect tool presented in the demo can be a complete disaster to test your application. In this context, the best practice is to demonstrate the benefits, limitations and constraints of the tool through a proof of concept.

In summary, you should choose the most important and complex tests that will be automated in your automation project and create all scripts using a demo or trial version of the automation tool.

As mentioned earlier, the demos present the key functionality of the automation tool through simple scenarios and tests. A proof of concept, in turn, aims to prove whether the tool will meet the real needs of your automation project.

Among the aspects that should be analyzed in a proof of concept, we should highlight:

- Assess whether the functionalities of the automation tool work as shown in the demo;
- Assess whether manuals are adequate and informative;
- Assess whether there are critical defects that prevent the use of the main features of the tool;
- Evaluate whether performance degradation or memory leakage occurs when the tool runs for many hours;
- Evaluate whether the tool recognizes the custom components and components (buttons, fields, menus, etc.) of the application that will be tested;
- Assess whether the tool and scripting language offered is robust enough to handle complex testing;
- Evaluate whether the tool is really easy to use and whether the tool provides resources for creating script libraries to facilitate reuse;
- Assess whether the tool works properly in the existing infrastructure/environment and assess whether any additional investment will be required to adapt the infrastructure/environment to the minimum requirements required by the tool;
- Identify tool limitations and restrictions to avoid false expectations.

Scale infrastructure accordingly

Many test automation projects fail due to undersizing the infrastructure. Automated testing tools require high-performance computers with fast processors and large amounts of memory. In addition, these computers must be dedicated exclusively to test automation (during testing execution the computer cannot be used for another purpose).

The computers where the tests will run must be similar or more capable of processing compared to the computers used to create automated test scripts, otherwise low performance issues will occur.

The execution of automated tests is very sensitive to external influences. We must reinforce that subtle differences in the environment (operating system, network, versions of programs, data etc.) may impair or prevent the execution of automated tests.

Planning in detail the infrastructure required to create and run automated tests is a key factor in the success of a test automation project. Among the aspects that should be considered, we should highlight:

- **High-performance computers**: The computers that will be used to create and run automated tests must comply with the minimum requirements required by the automation tool;
- **Dedicated computers**: Computers where automated tests will run must be dedicated to this function, otherwise tests may be impaired and performance degradation may occur;
- **Isolated environment**: The environment should be restricted to the test automation team to avoid destabilizing the integrity of the environment through external influences;
- **Controlled environment**: All the details of the environment (operating system, network, versions of programs, data, etc.) shall be planned and controlled, under penalty of harming or preventing the execution of automated tests;
- **Production-like environment**: The environment where the tests will be performed must be equal to or similar to the production environment, under penalty of finding false positives, that is, the tests are successfully executed in the testing environment, but the functionalities have defects in production;
- **Consistent data** mass: The data used in the tests should be known and representative, in addition, they should be stored in a baseline to allow it to be retrieved every time automated tests are run.

View test automation as a project

Test automation is a combination of testing and software development, after all, the test scripting activity is a pure programming activity. Thus, we can affirm that test automation should be seen as a project with its own characteristics, that is, it requires detailed planning, as well as design, development and testing activities, such as the development of conventional software.

As a rule, the creation of automated tests is performed by testers specialized in development, also known as Tester-Developer or Automated. This professional must have the developer profile and be properly trained to use the test automation tool.

Based on what has been exposed, experience has shown that the following aspects should be considered in test automation projects:

- The automation project should be seen as a project with its own characteristics, that is, it requires detailed planning, as well as project, development and testing activities;
- Test scripts may have defects, so it is recommended to use a bugtracker tool to manage these defects;
- Test scripts must undergo a Software Configuration Management and version control process;
- During the development of test scripts, best practices used in conventional software development projects should be applied;
- The Tester-Developer must have the profile and interest in working with test automation, not always a good tester or test analyst will become a good Tester-Developer;
- The human resources that will act in the test automation project must be adequately trained. The more skilled, the more likely the project will be successful.

Align expectations and ensure the collaboration of all involved

Test automation projects often fail because the people involved have different perceptions about the benefits and limitations of such a project.

Typically senior management believes that test automation is the solution to all problems. Architects and developers are unaware of the need to create mechanisms to increase stability during application design and development.

The testing team, in turn, often believes that test automation is a simple task that requires no training and is summarized only by transforming manual tests into automated test scripts through the recording capabilities offered by test automation tools.

These erroneous assumptions are the result of a lack of knowledge of the benefits and limitations of a test automation project. The advertising campaigns of automation tools, in turn, make this scenario worse because they sell an image that test automation is the solution to all quality problems. Among the most common marketing tricks used by companies that make automation tools, we should highlight:

- Develop quality software and increase your company's revenue and profitability through the XYZ automated testing tool;
- Create sophisticated tests with minimal training;
- Use the XYZ feature to write scripts and eliminate script encoding time;
- Increase productivity and decrease maintenance time by sharing scripts and libraries;
- Easily identify issues through the XYZ feature used in our reports;
- Use the XYZ feature of automatic object recognition and ensure that tests are run even when the graphical interface has changed.

In test automation projects, it is necessary to align expectations and collaborate between senior management, developers, architects, and testers. It is of paramount importance to explain in detail the objectives and scope of a test automation project and, above all, to ensure that the expectations of all stakeholders are realistic. In this way, we were able to eliminate folklore and myths created by the marketing strategies used by companies that make automation tools.

The recommended strategy to ensure that expectations are aligned with respect to the benefits and limitations of a test automation project should consider the following activities:

- Demonstrate the benefits, limitations and restrictions of the tool through a proof of concept;
- Plan in detail the infrastructure required to create and run automated tests to avoid undersizing issues;
- Involve developers and architects in the test automation project in order to incorporate mechanisms to increase application stability;
- Involve testers to align expectations in relation to the objectives and strategy of test automation, as well as to foster training and training;

- Involve senior management to align expectations regarding return on investment, benefits, limitations and constraints of a test automation project.

Test automation is a long-term investment

Test automation is undoubtedly a good practice in a software testing process. However, test automation is not a practice that is adopted overnight. As we mentioned earlier, the **need to automate testing will** come naturally as a result of the evolution of the maturity of the testing process.

Despite all the benefits of test automation, investments are high. Rex Black, software testing guru, in his famous article called "Successful Investing in Software Testing" presents a hypothetical scenario with the costs required to perform formal, manual and automated testing and the return on investment (ROI) obtained in each of these approaches.

In this article, Rex Black, part of the premise that the cost to fix a defect found in the development phase costs about $10, in the test phase it costs $100 and in production it costs $1,000, as can be seen in **Table 1**.

		Informal Test	**Manual Test**	**Automated Testing**
Investments	Staff	0	60.000	60.000
	Infrastructure	0	10.000	10.000
	Tools	0	0	12.500

	TOTAL	0	70.000	82.500
Development phase	Defects found	250	250	250
	Cost to fix	2.500	2.500	2.500
Testing phase	Defects found	0	350	500
	Cost to fix	0	35.000	50.000
Production	Defects found	750	400	250
	Cost to fix	750.000	400.000	250.000
Cost of Quality	Compliance (investment in prevention)	0	70.000	82.500

	Non-Compliance (cost to correct defects found)	752.500	437.500	302.500
	TOTAL	752.500	507.500	385.000
Return on Investment		**N/A**	**350%**	**445%**

Table 1. Return on investment from formal, manual and automated tests

In the example proposed by Rex Black, the approach using automated testing ensures a return on investment of 445%. The message is clear: test automation pays for the investment. The big problem is how much investment is needed and when that investment will start to pay off.

You may notice that rex black's example considers only a $12,500 investment for the purchase of the test automation tool. On the other hand, we should remember that a tool is only the means by which we implement test automation, but test automation is not limited to using a tool.

As we saw earlier, test automation should be seen as a project with its own characteristics, that is, it requires detailed planning, as well as design, development and testing activities, such as the development of conventional software. That is, the investment in test automation is not limited only to the costs of the test automation tool.

Unfortunately, **many test automation projects often fail because** senior management often believes that the only investment needed for test automation is the purchase of a tool. However, there are many other necessary investments, such as:

- Hiring or training of personnel to carry out the management of the test automation project;
- Hiring or training of personnel to design/create automated test cases;
- Purchase or improvement of infrastructure / environment to achieve the minimum requirements required by the tool;

- Expenses related to incorporating stability into the application to make it easier to test;
- Expenses related to the creation of proofs of concepts or prototypes.

Test automation is an investment with guaranteed return when applied correctly. However, the return is long-term, i.e. there is no immediate return.

As we mentioned earlier, test automation should start from a proof of concept and gradually grow to maturity through a set of robust and stable scripts that ensure adequate test coverage.

The path to maturity in test automation is tortuous and time-consuming. It takes a lot of persistence to achieve maturity in test automation. The best practice to increase the chances of success is to evolve itatively. Thus, the losses arising from a wrong decision in an iteration will be low and manageable.

Manual testing is irreplaceable

According to Cem Kaner, author of the book "Lessons Learned in Software Testing", the purpose of test automation can be briefly described as the application of strategies and tools to reduce human involvement in repetitive manual activities.

We must recognize that test automation should not be employed as a substitute for manual testing. It should be introduced as an additional technique whose primary purpose is to add value, without, however, invalidating the existing manual test. After all, manual and automated testing are different test approaches that reinforce each other.

Thus, through test automation, testers will be able to reduce their involvement in repetitive manual activities and focus on complementary testing approaches that require the application of intuition and judgment. Exploratory testing is an interesting complementary testing strategy since it makes intensive use of the tester's intuition and judgment.

Exploratory testing is, in its most basic definition, the creation and execution at the same time of a test. When performing an exploratory test, the tester typically does not have detailed information about what it will test and how it will test. The tester builds on his experience as well as on the knowledge he acquires about the application while running the exploratory test.

From this perspective, we can affirm that exploratory testing is an itenative and empirical activity of exploration that requires comings and goings in a continuous investigation process where intuition, creativity and experience of the tester are indispensable to ensure the efficiency of the test.

The great benefit of using exploratory testing is linked to its empirical nature. **Automated tests always repeat the same tests, the** same paths, the same operations. Through exploratory testing we have the opportunity to empirically and itterically create new tests and follow new paths, which in turn leads us to the discovery of new defects.

Conclusion

Test automation should not be used as a substitute for manual testing. It should be introduced as an additional technique whose primary purpose is to add value. The focus should be on improving the testing process used in your company. The need to automate testing will come naturally as a result of the evolution of the maturity of the testing process.

Finally, we must remember that test automation is an investment whose return is long-term. The path to maturity in test automation is tortuous and time-consuming.

In this journey, unforeseen problems, incorrect assumptions, unaligned expectations, defects in the automation tool, lack of stability in the application, problems in the infrastructure /environment, among other aspects discussed in this article will probably arise. The important thing is to learn from their own mistakes and move on, after all, those who cannot remember the mistakes of the past are doomed to repeat them (George Santayanna).