ROI Measurement for Test Automation Projects

A Case Study

Table of Contents

Abstract

In Agile software development, test automation has evolved as a critical component of the continuous delivery of software. As greater automation test coverage gives stakeholders more accurate information of stability of high-value business features of their application and business. This will, in turn, help them in making more informed decisions at the business level. To understand if test automation will be a success or not, ROI will help to get insights.

Though the business decision-makers agree that test automation is a good idea but would not commit appropriate resources as they are uncertain about the costs of automating and maintaining a test automation framework and scripts. There will also be no transparency on the budget required, timelines to expect, and the value that it will deliver.

To get a buy-in from all the stakeholder, ROI measurement will help where business decision-makers can better understand the value that test automation delivers over time, and technical team members would get insights into the effects of what is required, the parameters involved, test life cycle timelines, robustness, and maintainability of the test framework and scripts. With this, both categories of users can use ROI as a tool to drive right decision making, leading to highly efficient test automation.

This paper talks about the below:

1. Why should we measure ROI?
2. Parameters to be considered to measure ROI
3. How to measure ROI?
4. Steps to improve ROI
5. Case study on how we achieved it

Introduction

Return on investment (ROI) can be defined as a metric used to calculate the efficiency of any investment. If this is applied to test automation projects, it will depict how many cycles of execution is required to achieve the breakeven point and from when there will be positive returns from the investment.

This can also be computed before starting the project to understand the wait time to achieve breakeven point, and whether there are any aspects to improve ROI etc. Usually, regression scenarios would be automated as during regression testing testers would need to work with huge data sets and on multiple browsers etc. The test automation ROI helps in finding out whether the shift to automation is worth performing.

The effort for manual testing along with the parameters (discussed in detail in further sections) have to be compared with the effort taken with the help of an automation tool. Using these efforts, the ROI for both cases can be checked and a decision arrived at.

Why Should We Measure ROI?

Measuring anything will help us in identifying the efficiency and control it. Without measuring, nothing can be planned or controlled. Moving to Test automation ROI, it's a key performance indicator in understanding the profitability of the investment. Also, how and by when it reduces the cost as time progresses.

Calculating return on investment (ROI) on automation project will help determine:

1. A simple approximation of cost
2. The types of automation for the project
3. The tools required
4. Level of skill required for the testing resources for the project.
5. Whether effort that is put in is justified
6. The planning process for the project
7. The strategies to follow to maximize their return

How Should We Measure ROI?

Identify the activities related to Test environment setup, Test development, Testing, Maintenance phases. Identify the effort required for each activity.

Effort can be easily converted to cost by calculating the cost of each resource.

Using the formula mentioned, cost savings for each cycle can be computed.

Steps to improve ROI

Right scope: Identify the high-priority business scenarios that are repetitive tests instead of automating everything. The simplest way is to start with Smoke suite and then move to regression suite. Also, ensure that there is no complexity or dependency on other systems or hardware as automating and executing them might not always be feasible and may impact the estimates.

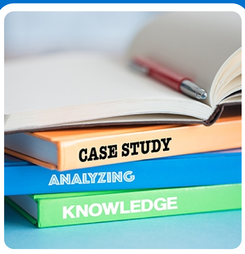
Right test design: All the tests that are considered for automation should be end-to-end scenarios that have good code coverage. Such scenarios would help in reducing the number of tests and time to execute.

- **Right test data:** The test data that is used is also critical. It is comprehensive and relevant that can match the end user's data. Different data sets can be created for valid, invalid and boundary conditions. This data should be verified before every major release to check its relevance.
- **Right plan for execution and maintenance:** Once the test scripts are created, there should be a proper plan to execute and maintain them on a regular basis instead of pushing to execute just for a release. If these are executed frequently, the issues can be found early in the cycle, code issues can be fixed, and code changes for the application enhancements can be done early which is critical in identifying defects in the application
- **Right coverage:** If the application has to be tested across multiple platforms or browsers or devices, it is advisable to select a cloud-based testing platform that provides different combinations of these This will help in reducing the cost of infrastructure, increase the coverage and improve test effectiveness
- **Right automation framework:** An automation framework is said to be an effective one if it provides the below features:

  Ease of use: Framework should provide a set of reusable keywords that would enable the engineer to consume them and write an automated script.
  Scalable: It should aim at being scalable and reduce maintenance efforts in a world of demanding business needs, evolving requirements, changing technology interfaces and people
  Cross browser testing: The scripts that are developed using this framework should work with all the leading browsers. User who has scripted with Chrome browser should be able to execute on Firefox or Internet Explorer with just change in Run settings.
  Data-driven testing support: User should be able to change the data easily without going into any technical details. Framework should provide an option to execute the same script with different test data inputs.
  Good and flexible reporting mechanisms: The execution reports should be easy to understand with complete details like the steps performed, test data used, the verifications points, and most importantly, screenshots for the failed steps.
  Build and use reusable components: User should be able to combine a few keywords and create a new reusable keyword that can be used in different test scenarios.
  E-mail notifications: Once the execution is completed, stakeholders have to be notified with an email that can include the details of the modules, test scripts, and run settings
  Provisioning of Batch Runs and Scheduling: User should be able to create batches with different sets of run settings or modules or test scripts and also schedule them to execute at a specific time
  Easily integrate with CI tools: User should be able to integrate the framework easily with any of the CI tools to achieve continuous testing along with the development.

Case Study

We have worked on a project where we started from developing the framework using open-source tools by integrating open-source external report into the tool. Over a period of 3 years team has developed 1600+ scripts and are maintaining them. These test scripts are executed for every sprint which is 14 days in duration. And these scripts are also executed for ad hoc release request.



Calculating ROI

Since ROI is a unitless number, it really does not matter whether the savings and investment amounts are in dollars or time. For ease in calculation, hours will be used because most of our inputs are in the form of time.

Before we actual start calculating the ROI, we need to consider all the facts/tasks which effects the ROI.

Calculating ROI

Since ROI is a unitless number, it really does not matter whether the savings and investment amounts are in dollars or time. For ease in calculation, hours will be used because most of our inputs are in the form of time.

Before we actual start calculating the ROI, we need to consider all the facts/tasks which effects the ROI.

Here we would like to segregate all facts/tasks in 3 different areas mentioned below:

1. Initial time for development setup
2. Actual Development time
3. Monitoring and Maintenance time

Initial time for development setup

The pre-phase of the development activity involves parameters like setting up the automation tool, setting up different environments, exploring the applications, time spent for initial data setup etc.

Initial Time For Development Setup			
Manual		**Automation**	
Tasks	**# Hours**	**Tasks**	**# Hours**
Time spent to setup infrastructure - Machines Setup	0	Time spent to setup Automation Tool	40
Extra time spent to setup infrastructure - Server / environment	0	Extra time spent to setup infrastructure - Machines Setup	4
Time spent to Explore and understand the application	96	Extra time spent to setup infrastructure - Server / environment	0
Time spent to setup the initial test data	8	Extra time spent to setup infrastructure - Continuous integration/repository	0
		Extra time spent to setup infrastructure - Cross Browser Setup	0
		Time spent to Explore applications/understanding manul TC	96
		Time spent to setup the initial test data	8
Time spent for the initial development setup	**104**	**Time spent for the initial development setup**	**148**

Actual Development time

In this phase, all the development activities will take place. Parameters like the average number of scripts developed per day and average time to develop each script, code push and sanity check, code review, etc., can be considered to calculate the actual development time. For a manual testing one can consider average scenarios authored per day, average time to author each scenario, total time to review the test case document etc. can be considered to calculate the actual development time

Initial Development Time			
Manual		**Automation**	
Tasks	**# Hours**	**Tasks**	**# Hours**
Average scenarios authored per day	40	Average sceanrio devloped per day	6
Average Time spent to author each scenario in Mins	12.00	Average Time spent to develop each scenario in Mins	80.00
Total time spent to author all scenarios	320.00	Total time spent to develop all scenarios	2133.33
Total time spent to review test scenarios	40	Total time spent to review codes and implement code review changes	270
		Time Spent to code push and sanity test	50
Initial Development Time	**360.00**	**Initial development time**	**2,453.33**

Monitoring and Maintenance Time

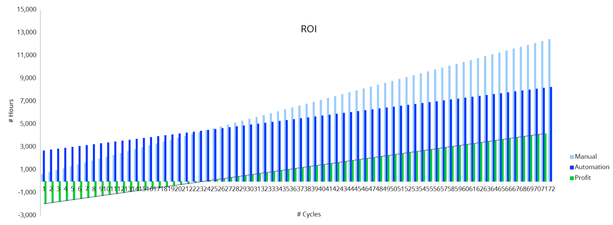
This is the phase where automation saves a lot of time compared to manual testing. All maintenance-related activities can be considered as a parameter to calculate the actual number of monitoring and maintenance hours.

Monitoring and Maintenance Time			
Manual		**Automation**	
Tasks	**# Hours**	**Tasks**	**# Hours**
Average scenarios executed per day	80	Total time spent to execute all scenarios	16
Average manual execution time per scenario	6.00	Total time spent to investigate Report	4
Time spent to execute all scenarios in Hours	160	Average automated execution time per scenario in Minutes	0.75
Time spent to report app failures and retesting in Hours per cycle	6.00	Average Time spent to report app failures and retesting per cycle	6.00
		Average time spent for dry run and data setup per cycle	8
		Average scripts requiring manintenance in %	6
		Avg maintenance time per failed test in Mins	10
		Average time to spent exploring application/user stories	7.00
		Avg % new scenario developed per cycle	1
		Total execution Time per execution cycle in Hours	20
		Total time spent to report app failures and retesting per cycle in Hours	8
		Total time spent for dry run and data setup per cycle	6
		Total maintenance time per cycle in Hours	16.00
		Total time spent to explore application/user stories	7.00
		Total time spent develop new scenarios per cycle	21.33
Total time spent for Monitoring and Maintenance per cycle	**166.00**	**Total time spent for Monitoring and Maintenance per cycle**	**78**

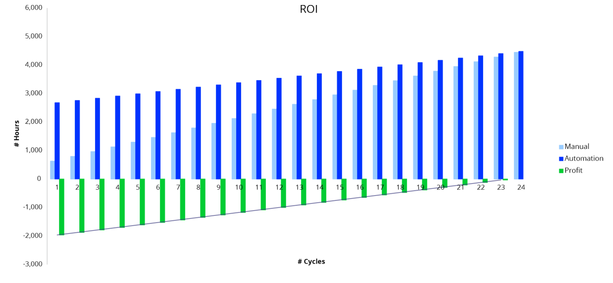
Cost Savings

Savings is the difference between the cost of running a set of tests manually versus running the same tests automatically several times over some period.

Over 3 years, 72 regression run cycles we gain around 2000+ hours when compared to manual testing



Let us consider another scenario, where after one year, the project ended or the test automation framework is abandoned; in either case, it would have been more cost-effective to go with manual testing.



Conclusion

ROI metric is important to get visibility on the success of test automation even before we start it. Hence this will help in making strategic decisions.

Get in touch with us for feasibility studies, joint workshops, pilots & ideation sessions. **Let's connect**

Contributors

Geetha Pavani Achutuni
Senior Manager, Project Delivery
geetha.achutuni@acsicorp.com

Sunil Kumar
Principal Software Engineer, Project Delivery
sunil.kumar@acsicorp.com