
# TEST AUTOMATION IN THE AGILE WORLD

## Abstract

Agile project management has recently been picking up steam in the management world. It has, to a large extent, convinced non-believers of the benefits of agile practices - the singular most significant being the reduced time to market. Let's face it, in an ever changing world, how else are we to survive unless we are centered on the balancing stone of agility? However there are still a few skeptics out there who are as yet unconvinced that all types of projects can adhere to this methodology. Test automation falls into one such bucket where the traditional agile principles cannot be fitted in as-is. This paper aims to put to rest qualms about being able to practice agile in a test automation project execution. We start off by touching upon the delivery goals behind a test automation project and then move on to dig deeper within to expose the hidden challenges that lie await. We subsequently address the challenges with simple strategic solutions that are viable in most environments, all the while aiming to re-instill faith in the 'Agile' way.

HARMAN

## Introduction

**The goal of this paper is to:**

A) To showcase a typical engagement of automation of regression test cases

B) To discuss the challenges of test automation and solutions to address them through simple agile practices and process methods.

C) To elucidate a Life Cycle Management that brings together all the answers discussed.

D) To go over strategies to showcase the value delivered through automation

## Automation of Regression Test Cases

A test automation project more often than not consists of a pre-existing system/application for which manual scripts may or may not have been written. Manual scripts is the term generally used to describe test cases written by a test engineer who intends to execute the test steps manually to perform unit, integrated or regression testing of a given application. The system itself could have been built in a non-agile mode or be a legacy system but stable enough to derive benefits of automation. The general goal of the project is to automate the manual scripts so that they can be run at will and thereby free up the resources' time to divert to other areas.

**The benefits of automation are quite obvious.**

- Firstly automated runs speed up the testing cycle thereby ensuring that the application is available in the least possible time.
- Secondly given that there is only a bare minimum cost involved in running the automated scripts, quality of the application is in no way compromised.
- Thirdly, the most evident benefit is the cost savings to the organization as a whole. For large applications with a wide user base, investing in such an effort is a natural choice and a vital part of any CTO's vision.

From a delivery standpoint, the very crux of every project is to deliver value. What is of business value to the client must be very clearly understood when it comes to an automation program and more so when we, as the vendor, are asked to drive it. The critical step is to familiarize ourselves with the pain points they are trying to address. For one of the clients where we were fortunate to have served, the company was releasing a major release to its clients. Given the size of the application which was in development for over 10+ years, they were looking for automated test cases that could perform regression. And there we were - with no domain experience, no application experience and no product knowledge, embarking on an agile journey and what we uncovered and overcame with the agile spirit to make it a success is what follows.

## Challenges in the Automation Space

There are specific challenges within the test automation space that require careful consideration before we commit to the agile mode of project execution.

## How do we define Scope?

The Problem - The requirements themselves are a set of Test cases. The stakeholder will look at 'more bang for the buck' – read as - the optimal automation code is the one that captures the most defects consistently, robust enough to be run multiple times and dependable enough to ensure ongoing cost savings. With a tag line like that how does one guide the stakeholder by asking the right questions to help define the scope?

The Answer - For a huge application, it's quite expected that there's a mountain hill of test cases waiting. There are 2 options to approach the scope factor.

Option 1: One way to approach is to assess the history of test case failures and then prioritize accordingly. This however implicitly assumes that the data is accurate and complete. In addition it's possible that a lot of these test cases are new and although critical may not have been run often enough to get into the right priority.

Option 2: The safer approach is to ask the client for a prioritized list of modules/sub-modules they deem critical and start with those test cases first. This assured that the right test cases were picked and that 'more bang for the buck' sentiment was addressed in parallel.

Another good idea is to consider Lisa Crispin's 4 Agile Testing Quadrants. These quadrants take into account the purpose of the testing by bucketing it into:

Quadrant 1: Technology-facing tests that support the team

Quadrant 2: Business-facing tests that support the team

Quadrant 3: Business-facing tests that critique the product

Quadrant 4: Technology-facing tests that critique the product

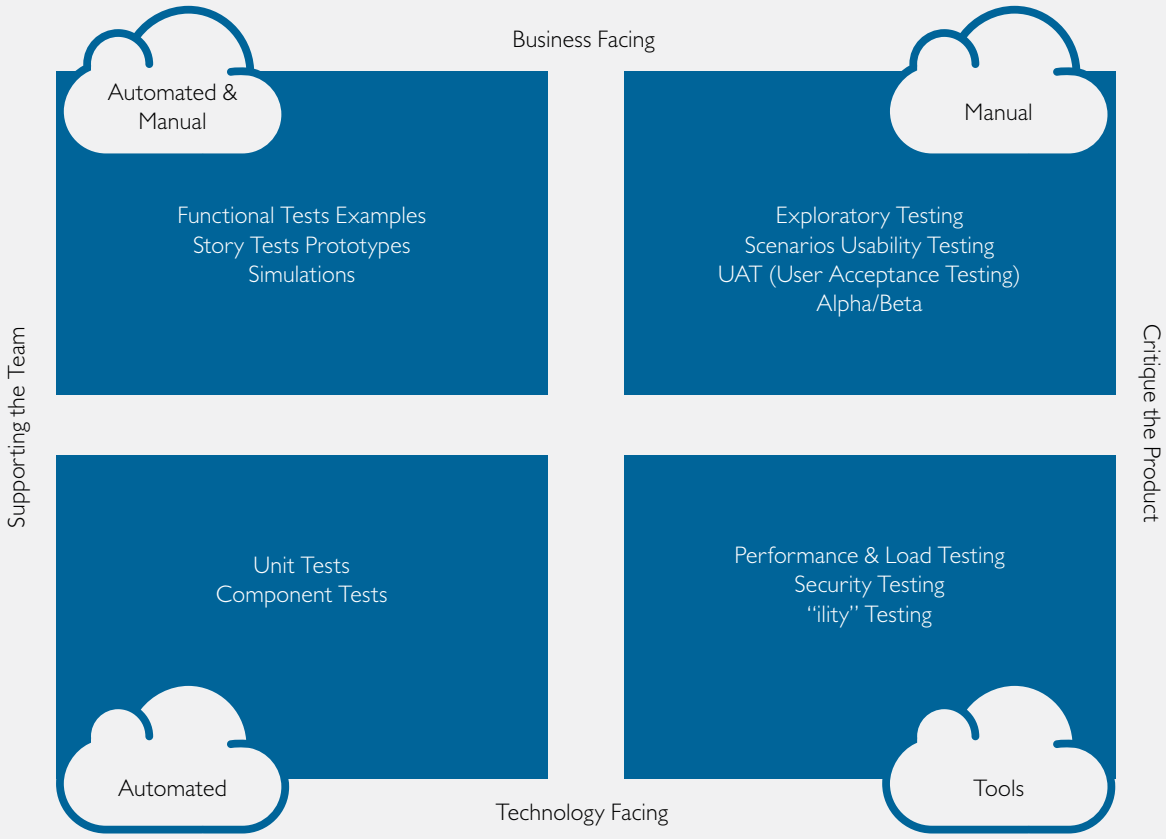


**Figure 1: 4 Quadrants of Agile Testing**

In alignment to this theory, we should ideally pick test cases that fall into quadrants 1 & 2. Following this methodology will require collaboration between the stakeholders and automation engineers and bring about sanity in an otherwise chaotic selection process. This will also ensure that stakeholder opinion is heard and accounted for during scoping.

## How do we Estimate?

The Problem - A test case could be estimated in a couple of different ways. The easiest of these would be to estimate based on the number of test steps. The catch here is that since these test steps are written manually, the complexity of each test step may not be the same. Consequently again each test case of the same grouping may not be of the same size/estimate. It is also not possible to immediately assess whether a test case is automatable just through a round of discussions since a BAs knowledge of the automation framework is limited and the automation engineer's knowledge of the functionality is just as limited. Especially in the first few sprints this topic manifests into the 'elephant in the room'.

The Answer – When you are at the stage of estimation, it is assumed that the test case is now a story i.e., it has been groomed and scrutinized for automation feasibility. Typically the size of a test case is usually approximated based on the number of test steps. This is wrong for various reasons. Firstly each step could be a conjunction of multiple steps or possibly a full blown scenario. Secondly it's highly likely that the all the test cases are not written by a single tester and therefore varies in terms of clarity and complexity of each test step. Thirdly, there could be hidden pre-requisites of a step that require more implicit work to get them automated.

The following approach for estimation has been executed in HARMAN and is now a proven execution approach. At the outset, the discovery phase of the project was involved in automating 3 test cases from the largest modules chosen by the client. We then came up with an estimation criteria list as indicated in the figure below while mapping a specific range of values of each criteria to a T-Shirt size value of XS, S, M, L, XL and XXL.

| AUTOMATION ESTIMATE | | Testcase Complexity Definition (TCD) | | | | | |
|---|---|---|---|---|---|---|---|
| S.No | Esimate Criteria | XS | S | M | L | XL | XXL |
| 1 | Number of Execution Steps | 1 <=15 | 30 | 45 | 60 | 75 | 90 |
| 2 | Number of Validation Steps | 1<6 | 12 | 18 | 24 | 30 | 40 |
| 3 | Manual Execution Time | 2 <=7 min | 14min | 21min | 28min | 35min | >=42<=50 |
| 4 | Number of Complex Controls | 1 <=2 | 4 | 6 | 8 | 10 | >=12<=50 |
| 5 | Number of DataRows | 1 <=5 | 10 | 15 | 20 | 25 | >=30<=40 |
| 6 | Number of User Roles | 1 | 3 | 5 | 7 | 9 | 11 |
| 7 | Number of Repeated Steps(Same Steps) | 2<=7 | 14 | 21 | 28 | 35 | 42 |
| 8 | Number of External tool Steps | 0 | 1 | 2 | 3 | 4 | 5 |
| 9 | Number of Pre-requisites Steps | 1 | 2 | 3 | 4 | 5 | 6 |
| 11 | Explicit Delay between Test Steps | 1 min | 2 min | 3 min | 4 min | 5 min | 6 min |
| 12 | Number of Dynamic Controls | 1 | 2 | 3 | 4 | 5 | 6 |
| 13 | Require Automation Framework change | 2<=4Hrs | 8Hrs | 12Hrs | 16Hrs | 20Hrs | 24Hrs |
| 14 | Number of steps which need more Investigation | 1 <=2 | 4 | 6 | 8 | 10 | 12 |

**Figure 2: Estimation Criteria for Discovery Phase**

Additionally each T-Shirt size and criteria combination will have an estimated number of hours as shown below.

| AUTOMATION ESTIMATE | | Testcase Complexity Weightage (TCW) | | | | | |
|---|---|---|---|---|---|---|---|
| S.No | Esimate Criteria | XS (Hrs) | S(Hrs) | M(Hrs) | L(Hrs) | XL(Hrs) | XXL(Hrs) |
| 1 | Number of Execution Steps | 10 | 18 | 26 | 34 | 42 | 50 |
| 2 | Number of Validation Steps | 4 | 8 | 12 | 16 | 20 | 24 |
| 3 | Manual Execution Time | Not Applicable to estimate script development | | | | | |
| 4 | Number of Complex Controls | 2 | 3 | 4 | 5 | 6 | 7 |
| 5 | Number of DataRows | 0.5 | 1 | 1.5 | 2 | 2.5 | 3 |
| 6 | Number of User Roles | 0.5 | 1 | 2 | 4 | 6 | 8 |
| 7 | Number of Repeated Steps(Same Steps) | -2 | -4 | -6 | -8 | -10 | -12 |
| 8 | Number of External tool Steps | 0 | 1 | 1.5 | 2 | 2.5 | 3 |
| 9 | Number of Pre-requisites Steps | 0.25 | 0.5 | 0.75 | 1 | 1.25 | 1.5 |
| 11 | Explicit Delay between Test Steps | 0.25 | 0.5 | 1 | 1.5 | 2 | 2.5 |
| 12 | Number of Dynamic Controls | 1 | 3 | 5 | 7 | 9 | 10 |
| 13 | Require Automation Framework change | 4 | 8 | 12 | 16 | 20 | 24 |
| 14 | Number of steps which need more Investigation | 3 | 6 | 9 | 12 | 15 | 18 |
| **Automation Time (in hours) - Weightage Observation** | | **23.5** | **46** | **68.75** | **92.5** | **116.25** | **139** |
| **Automation Time (in days) - Weightage Observation** | | **3.92** | **7.67** | **11.46** | **15.42** | **19.38** | **23.17** |

**Figure 3: Sample T-Shirt to hour mapping**

From the above sample mapping it's evident, that a TC could take a minimum of 4 days and a maximum of 23 days. These values are fine-tuned over the discovery phase to arrive at the near accurate values. Below table gives the values for a hypothetical TC.

| AUTOMATION ESTIMATE | | 1 | |
|---|---|---|---|
| S.No | Esimate Criteria | Sample TC | |
| | | TCD | TCW |
| 1 | Number of Execution Steps | 50 | 34 |
| 2 | Number of Validation Steps | 8 | 8 |
| 3 | Manual Execution Time | 30 min | 0 |
| 4 | Number of Complex Controls | 1 | 2 |
| 5 | Number of DataRows | 1 | 0.5 |
| 6 | Number of User Roles | 1 | 0.5 |
| 7 | Number of Repeated Steps(Same Steps) | 11 | -4 |
| 8 | Number of External tool Steps | 0 | 0 |
| 9 | Number of Pre-requisites Steps | 1 | 0.25 |
| 11 | Explicit Delay between Test Steps | 2 | 0.5 |
| 12 | Number of Dynamic Controls | 0 | 0 |
| 13 | Require Automation Framework change | 0 | 0 |
| 14 | Number of steps which need more Investigation | 0 | 0 |
| **Automation Time (in hours) - Weightage Observation** | | **41.75** | |
| **Automation Time (in days) - Weightage Observation** | | **6.96** | |

**Figure 4: Estimation for a sample Test Case**

Once a sizable number of test cases are estimated in this fashion, similar test cases can be estimated in a poker planning session since now you have the groundwork done and the foundation laid.

## What is the definition of done?

The Problem - With an automation project deciding on the 'definition of done' can be quite daunting. While the automated test case can run beautifully how do you convince the client its doing what it's supposed to do – catch bugs, run though all test steps correctly and if the former mentioned criteria is met, how does one quantify the business value if there is no release planned to prove it's worth… In addition there are also challenges pertaining to what you should be showcasing in a demo and if it's a broken up test case how do evaluate its done?

The Answer – This can probably help with the definition of done in such a context -
"A test case is considered done when it is automated, code reviewed (internal or external), all defects closed and run for a specific number of times consecutively (either by client QA or a batch run process)."

This needs to be clearly called out in the engagement details to as to avoid questions during the course of the project. This statement indicates that the development of the test case is completed, executed by a stakeholder for correctness and regressed through multiple rounds (a set number) to ensure robustness. All parameters being met, it is marked done and accepted!

## How do we handle defects and rework?

The Problem - Defects in automation can come from either the application, inaccuracy in the test script or the automation code itself. This requires the collaboration of possibly 3 different teams (Development, Manual Tester and Automation Developers) to identify the owner for the defect itself. In short sprints where the focus needs to be on automation how does one manage such situations and allocate time for it?

The Answer – One of the 3 scenarios listed below should trigger the process of defect creation –

- Test case logs do not match the Test case Steps
- Test case fails on execution (and is verified that it is not due to an application bug)
- Test case does not give consistent results on consecutive execution

Any concerns of the test case post the 'code review' status by default falls into the category of rework.
Some realistic examples are:

- Automation changes after the 'Done' state
- Test case tagged incorrectly as 'To be automated'
- Test case steps/scenario is incorrect

Rework essentially means that the test case will be counted towards velocity and included in metrics. Some buffer time could be allocated as part of this towards the end but the rework will be accounted for as an additional test case. However it is always advised to keep a track of such re-work and define acceptable limits if at all it is to be absorbed as part of an agile goal.

## What do we do if the base application changes?

The Problem - If a test case that's written becomes invalid because of an application change, does it fall under rework? Does it even count towards velocity? What if it changes while it's in the middle of the sprint? What if it changes post automation and accepted but the client sees no value to it?

The Answer – The answer to this question actually lies in a crucial phase called scrubbing.
Scrubbing – Even before we step into grooming, it's prudent to have a scrubbing process where manual Test Engineers validate the test case for authenticity, validity and need for automation. This safeguards against to-fro between the automation engineers and clears the slate right out for Grooming.

Since scrubbing is done before automation it also ensures that the test case picked is actually agreed upon by the client. If a test case should no longer become valid due to an application change post acceptance then it is considered as a brand new test case and added back into the backlog. Needless to say, it will be considered for velocity like any of its peers. If it should change during the sprint, it would not count for velocity and the dip in velocity will force deliberation against prevention of such occurrences in the future.

## Life Cycle Management

It is imperative for one know at what stage a test case is in and how much more is left to do to reach a goal. With this context, the team at HARMAN came up with a set of stage gates depicted in the following figure.

| Gate | Name | Description | TC Count |
|---|---|---|---|
| **Project numbers to date:** | | **Current Stage Gates for all Automation Test Cases - to date** | |
| 1 | New | Scrubbed Test Cases submitted for Automation by the Client Manual team | |
| 2 | Analysis | Test Cases in pre-automation analysis | |
| 3 | Inquiry | Test Cases with analysis questions/responses | |
| 4 | Analyzed | Test Cases that have completed analysis and are ready for automation | |
| 5 | Coding | Test Cases actively being coded | |
| 6 | Code Review | Test Cases submitted for Code Review (Delivered) | |
| 7 | Certify | Test Cases submitted for Certification | |
| 8 | Certified | Test Cases that have passed all reviews and are waiting to be moved into the Batch | |
| 9 | In Batch | Test Cases have run in a regularly scheduled batch three or more time (Completed) | |
| 10 | Rework | Test Cases who's work has been halted due to rework/maintenance/blocking defect | |
| 11 | Retired | Test Cases removed from use | |

**Figure 5: Stage Gates**

A 'scrubbed' test case is first marked 'New'. If the number of test cases in the new status are less than the velocity of the next 2 sprints, then we need to flag a concern on 'Test Case Availability'. When an automation engineer picks it up for 'Automation Feasibility' or 'Grooming', it is moved into the 'Analysis' state. If there are questions surrounding the test case that need to be answered it is moved into the 'Inquiry' state and assigned to the client QA engineer. Once answered the automation engineer moves it to the next stage gate of either 'Analyzed' if all's well or 'Retired' if it cannot be automated. 'Analyzed' test cases are picked up for a given sprint and once an automation developer start coding it is moved into the 'Coding' stage. Once complete, it is moved into the 'Code Review stage'. The test case remains in this stage until code review comments are incorporated. Once done it is moved into the 'Certify' stage where it is run for a specific number of times and the logs compared for any anomalies. Defects are likely to arise at this stage at which point it will be moved back into the 'Coding' status. Once it passes the 'Certify' stage, it goes into the 'Certified' status which is one step before adding into a batch for use by the client. The stage gate 'In Batch' aims to indicate how many test cases developed by the automation team are actively being used by the client. This detailed table was circulated to all concerned stakeholders as part of a status report, the interval of which was pre-decided at the start of the engagement.

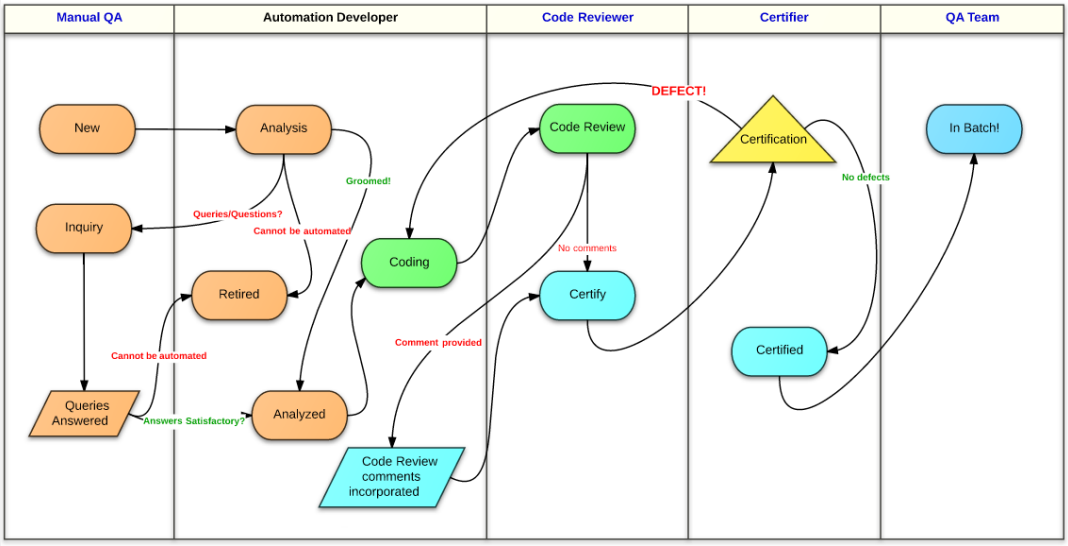


**Figure 6: Sprint Life Cycle**

A Sprint velocity goal is met if the planned TCs is submitted for 'Code Review'. The reason why the 'Code Review' and defect process is not accommodated as part of the sprint is to ensure that the completion focus is given to automation. Sprint 'n'(marked in green above) should account for the fixing of code review bugs of Sprint 'n-1'(marked in blue) and with this in mind 10% of the team's productivity per sprint is allocated to this task and capacity is adjusted accordingly. All tasks marked in brown should ideally happen in Sprint n-2. To recap, 10% of the team's productivity should go to getting future test cases into Analyzed state, 10% towards defects of the immediate past sprint, and 80% towards current sprint activities (coding, agile sprint ceremonies, etc.)

In addition to following the life cycle diligently, it goes without saying that the burn down charts in terms of number of test cases delivered against what's committed or planned as part of the end goal be published on a weekly basis to communicate progress(positive or otherwise). The aim is to give a good indication of the health of the project as a whole.

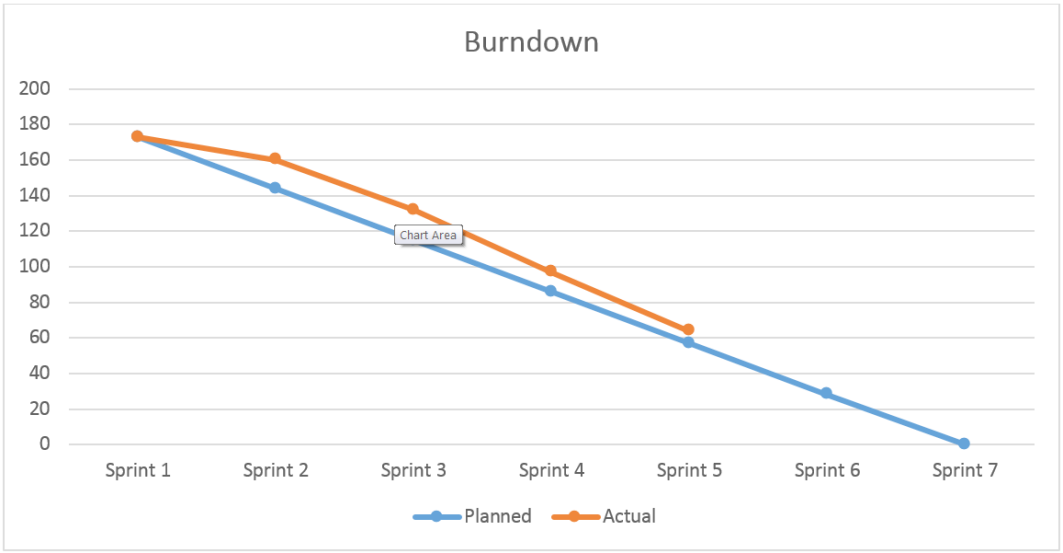


**Figure 7: Sample Burndown chart**

Given the multiple stages of Scrubbing, Grooming and Automation Feasibility where non-potential test cases are weeded out, there has to be acceptable buffer of test cases picked out for scrubbing which is in fact the starting point of the entire life cycle. 20% is a good starting point and can be increased post review over the completion of every 2-3 sprints. In addition, if the projected velocity of a Sprint is 20, there should be a minimum of 24 (i.e., 20 + (20 % of 20)) scrubbed test cases available for automation 1 Sprint in advance. This is to enable the team to have sufficient time to groom the test cases for feasibility.

## Projecting Business Value

While it is quite obvious that the strategic goal of the program is to ensure business value, it is just as essential to articulate it periodically to stakeholders on where we stand in their journey towards it. This can be projected in two broad areas – savings in terms of time and cost.

In order to project such data there is a need to assimilate the data of the current process and the time and cost involved. Let's say hypothetically that the client has handed over 3 modules of about 120 test cases to be automated. Following could be a list of questions asked in this regard.

- What is the time period required to execute these test cases manually?
- How much effort is required by the manual QA to do the same?
- How much of the total test time is allocated to these modules?
- What is the process in case a defect is found in any of the test cases in these modules?
- What is the effort /time involved for regression and integration runs?

A client's hypothetical Answer: "These 3 modules require 3 weeks of testing by 2 Manual QA. This is 30% time of the entire Test Cycle time. When a defect is raised and fixed, selected test cases are run based on the dependency matrix for the purposes of a sanity check. Regression and Integration runs cover a larger test suite for these modules, however the test cases being automated constitute about 40% of those test cases. The total time taken for a regression suite is about 2 months."

So in a non-automated scenario, the time taken for these 3 modules would be

3 modules * 3 weeks * 2 QA * 8 hours a day = 144 hours or 18 person days of effort

Let's assume that for the purposes of this discussion that we were able to automate 100 of the test cases and it takes about 20 hours to run for each module. The total time taken post automation is therefore

3 modules * 20 hours = 60 hours or 2.5 days (since automation can be run all day long)

This clearly indicates that we save 15.5 days of manual effort even if we consider 2.5 days of QA effort just to monitor the automated runs!! We first can draw up the graph to indicate the test time savings this brings about in a regular test run like so indicating a whole 28% of time savings overall!

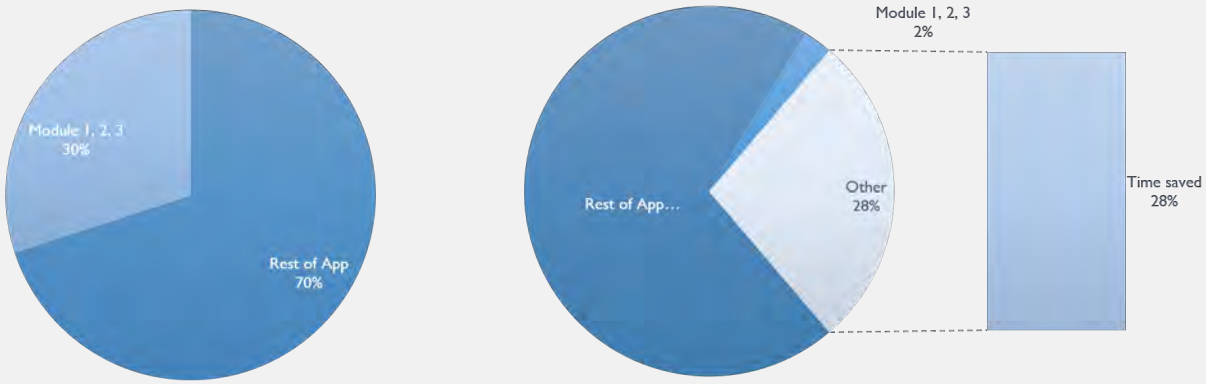


**Figure 8: Time Savings depiction**

These savings in time could actually give more time to developers for additional features to be added or could just concentrate on the stability of the system. Based on the stakeholder's business context, a strategic road map for this time savings could be mapped out or an earlier Release if that benefits the end users.
An additional area of interest is the cost savings automation brings in. For a 100 test cases run, let's say we figure out that there an average of 20% failures (if the application is stable there could be much lesser failures). If each failure required about 30mins of evaluation (this being on the higher side) from a manual QA to inspect the cause of failure, we are looking at 10 hours of effort. So total effort required for one cycle of testing could be summed up as -

| TASK | TIME REQUIRED (IN PERSON HRS) | COMMENTS |
|---|---|---|
| **TESTING EFFORT** | 1 | To schedule the automation |
| **REPORT INVESTIGATION** | 10 | Assuming only failures are investigated |
| **COMMUNICATION** | 2 | Interaction with the Dev team |

Taking an average cost/hour calculation of about $80/hour, and assuming these test cases will need to be repeated again for another 2 times to complete the entire QA cycle, a potential graph could be put forth like the sample below indicating that for one cycle this will cost $14,400 while with automation the spend comes down to $960. Similarly including the 2 regression cycles the cost pre-automation would be $43,200 while post automation, it's a more affordable $2,800.

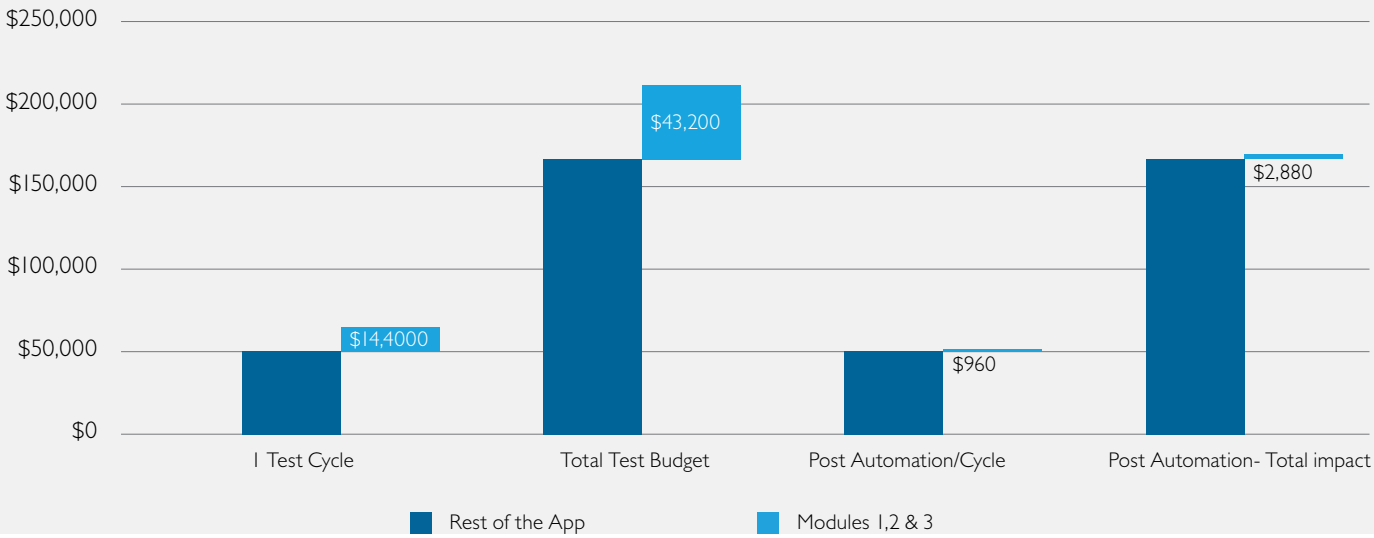


**Figure 9: Cost savings depiction**

This aim of this graph is to show the cost savings from a single cycle and the overall test execution process. These cost savings do not account for the management costs, infrastructure cost etc., but rather only based on the effort cost saved through automation. It also does not include any automation effort if in any possibility the failures are due to automation defects and not actual application defects, and must be called out explicitly. In any case it maintaining a small automation team for maintenance should be considered since as the application changes, so does the need for new test cases or re-writing of existing ones.

## Summary

The intent of this paper was to cover the agile concepts in a manner that can be applied to automation projects and in doing so answer all the challenging questions that were focused on in the early section of this artifact which cover the areas of

- Scope
- Estimation & Planning
- Execution Life Cycle
- Metrics that can showcase value

The methodology has been tried and tested, giving new faith in the way forward towards greater agile implementations. We, at HARMAN, hope this will give a head start to those who are seriously considering project execution in this context.