Better Testing Through Process Automation

Rick Tracy

(Hapalion Consulting & QualityMinds)

Hapalion@gmail.com

1 Abstract

Automating testing is currently still one of the top priorities among CIOs, and it remains a popular topic in the industry. Yet though it's not a new concept applying seems to still be in its infant, chaotic stages, and this is causing distress for both managers and practitioners of testing. This paper will explore some of the underpinnings of this dilemma and propose a new way forward, namely thought automating the processes surrounding testing instead.

We will take a short look into one large financial company's attempts to automate their testing for their data warehouse, and explore what went wrong and what was learned from it. Exploring the theory behind the automation is an important step on that journey, and so both the mentality and the application of that theory will also be presented as it was used by our team. Lastly we will look at where this led us to and why we believe this is a better way forward for testers and development teams as a whole.

This topic is still highly theoretical, with few companies in our experience moving towards this new paradigm. New tools are being created to facilitate it however and the more practical examples we see and testing minds involved we have the more refined we can make this. We look forward to spirited discussions and quality minded observations at PNSQC and beyond.

2 Biography

Rick is a freelance Tester and Scrummaster working in the finance and medical sectors and with Quality Minds and Hapalion Consultancy.

These positions have given him a variety of experience with different international audiences for test processes and reporting and plenty of experience doing it with more and less effective results. Rick's work passions lie in technique, optimization and communication, and occasionally the optimization of communication. He enjoys designing training programs and teaching and has focused most of his lectures on how to communicate ideas between different stakeholders and perspectives.

While he has a reputation for always having a story to tell, Rick prefers an interactive lecture or debate to a chalkboard presentation.

3 Introduction

Let's get this out of the way first: automation is not automatic. Regardless of whether you are changing a verification check into a test step, a manual activation into a triggered one, or a communication form into a report, creating automation takes work. Every moment spent automating is one less spent testing, so seeing it as an investment is advised and prioritizing accordingly is a must.

That said, we are in an age of ever-increasingly complex systems, products, and organizations, and we simply do not have the time to look over every aspect manually. We also have better and easier to use tools than ever before to aid us in exploring and testing items. So ignoring automation is something we cannot do.

This paper is a summation of what I've learned throughout my time going from 100% manual testing to figuring out what automation style works best with my work. It will cover the basis for automation, the arguments for and against, the balance needed to achieve it and the lessons learned as we created it. However, this paper seeks to go one step further and advocate for a different form of automation, one that focuses more on the strengths of both man and machine and how to utilize each in the best possible manner.

4 The Story So Far

To begin, I'm going to take you back to the days of 2012, where my foray into automation began. I was working for an international bank in the back office, where we processed around a million or more transactions per day through a global financial system pulling in records every minute from all around the globe (or the four corners of the world, for flat earth enthusiasts). This, as you may imagine, requires quite a large system of inputs, checks, databases, checks, reporting flows, checks, and tests. And that's just running it day to day.

We primarily focused on baseline cases in our team, but we were surrounded by six other project teams working on the exact same system and structure, meaning anything they altered would impact us and vice-versa. As you can imagine, this create quite the Wild West scenario, not only for architects and requirement engineers but also developers and testers. Every three months we were requested to put out a release of new inbound streams, new reports, and new functionality within the datawarehouse and to keep it all at the highest form of quality we could manage. Given that we were seven teams all depositing code in various states of readiness into one package, that highest form was not nearly as high as we wanted it.

After a while enough of us were upset about the quality degradation. Gathering a team of testers, developers, and business analysts we created our very first Agile Development team and set to work on what would later be called the Integration Quality team. Step one was to figure out how to bring a huge system with voluminous data into a reasonable scope for analysis. One brainstorm session later and Milestones was born.

Milestones was a Table Data Comparison tool built entirely from scratch using APEX oracle inputs and triggered commands. It was also our first foray into automation, as we found we could easily turn

collecting data into comparing data. The result was a tool which could read each table in our system and compare the impact of any introduced changes. While we had to load our tables manually, and define each table with the columns we wanted to compare, this did free up time and effort in making these queries manually and running them on volatile and ever-changing test environments.

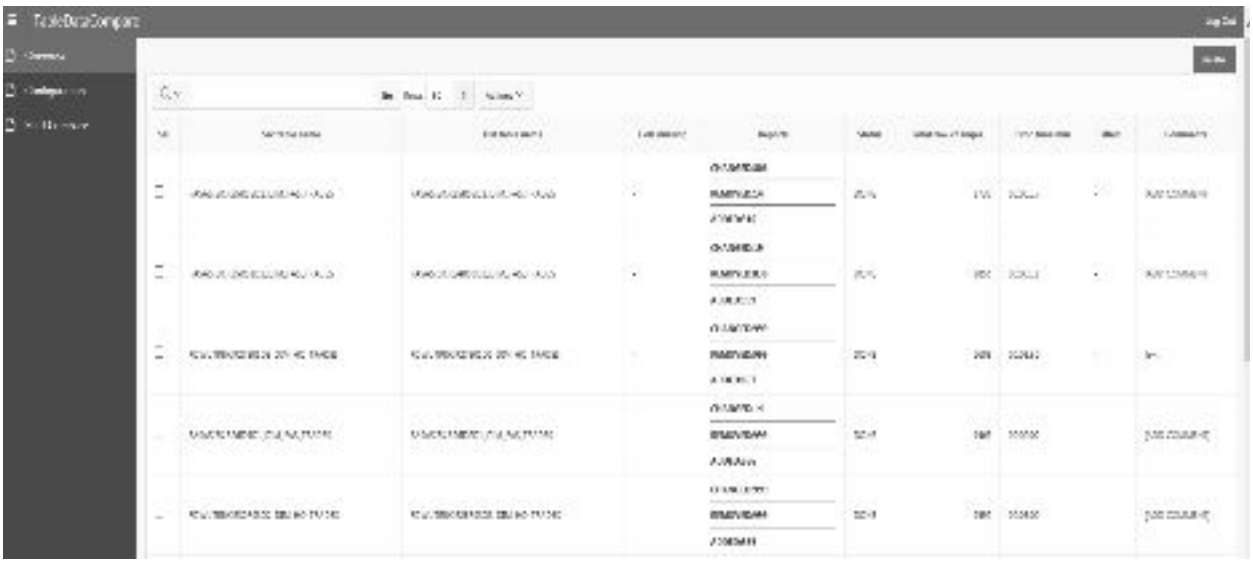

Image 1: Milestones Table Comparison tool

This tool sped up our analysis considerably and since it was used by a single team across teams we had a potential solution for many of our top issues in quality. However, once testers from each team began using it, flaws were discovered which almost tanked the whole project. Table configurations were still different per team depending on the changes they introduced, environments didn't match, and worst of all, the comparison done from Milestones only showed that there was a difference, not what caused it or whether it was expected or not. Trying to add useful manual checks to the program caused a significant slowdown, so many ended up using both, costing more time.

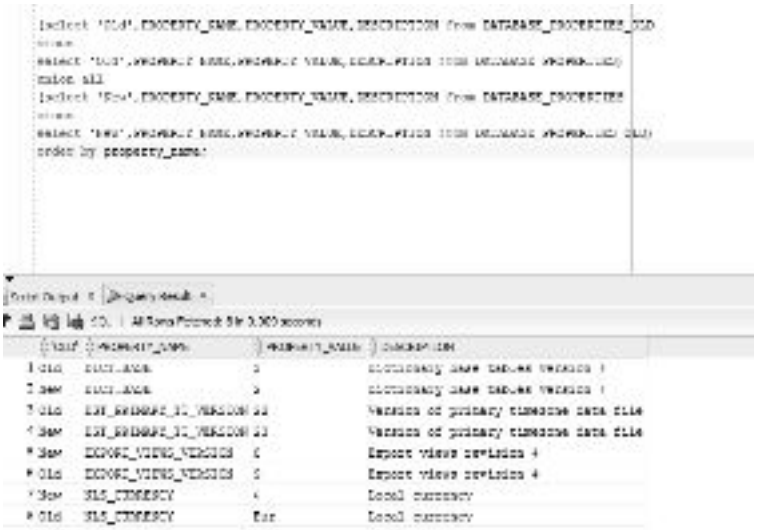

Image 2: Example of a manual script

Trying to turn a comparison program into a tester for integration tests was not working on this scale. Many testers started looking for additions to the program or designing their own. What could have turned into a failure instead set off a chain of innovation, and soon testers were coming with unique solutions to familiar problems.

One of those was a reset protocol for all environments, which would use a single repository of table settings from the Integration Quality (IQ) team to align any test environment in minutes. This script could be run manually or kicked off from Milestones once integrated. It didn't slow down the comparison process and solved one of our most insidious issues of alignment. This was immediately added and implemented into our process, and testers used it gleefully and often.

```
delete from TABLE_1;
delete from TABLE_2;
delete from TABLE_3;
delete from TABLE_4;
delete from TABLE_5;
delete from TABLE_6;
delete from TABLE_7;
delete from TABLE_8;
delete from TABLE_9;
delete from TABLE_10;

Insert into TABLE_1 (select * from BKP_TABLE_1);
Insert into TABLE_2 (select * from BKP_TABLE_2);
Insert into TABLE_3 (select * from BKP_TABLE_3);
Insert into TABLE_4 (select * from BKP_TABLE_4);
Insert into TABLE_5 (select * from BKP_TABLE_5);
Insert into TABLE_6 (select * from BKP_TABLE_6);
Insert into TABLE_7 (select * from BKP_TABLE_7);
Insert into TABLE_8 (select * from BKP_TABLE_8);
Insert into TABLE_9 (select * from BKP_TABLE_9);
Insert into TABLE_10 (select * from BKP_TABLE_10);
```

Image 3: Example reset script

Next step was to get the comparisons to give more direct and specific results, rather than a bland comparison. In order to do this testers looked into how they tested their specific cases, and we went for a batch/report generator validation automation style. This means taking batch commands and adding them to the validations, so that we could put both into Milestones and run specific cases. This combined both the comparison function and the reset function and used validation rules we injected to decide if the next step would turn on or not. This was our first step towards what eventually would revolutionize how we did automation.

5 Automation in Theory

Take a look at the following two pictures and see if you can detect the differences:

Image 4: Comparisons

Most of you will see the airplane, the time and the number on the bus first. Some will also see the missing head of the light or the color of the shirt of the walking man on the right. These are not the only changes, but they are the ones we see without taking the picture pixel by pixel.

The theory behind automation in comparisons is that a computer can do that low-level comparison by taking each component (in this case, the pixels) and comparing them, noting differences. This is a solid conclusion; Milestones did as much with any table comparison we had it do. So why don't we automate every comparison?

Mainly it has to do with what you see first. Most people looking at this picture will not look for any and all changes. They see a sky, a clock, and a bus. We are drawn to the important aspects of the picture, the ones which will have the most impact. So we immediately see a plane has entered the sky, the time has changed and the bus number is altered. Major impacts on major features.

Automated comparison will find things that a tester doesn't on first glance, but it will not prioritize like we do. It will tell you everything, and you have to tell it what's important. So while it is useful in a 100% coverage of comparison, it is slow and cares about every alteration equally. In order to automate a comparison to make it like a professional did it, we have to program priority, impact, features, and all other things we as people do automatically on first glance. That's a lot of extra work.

This shows us that people have a good and easy grasp of the priorities and features while computers have a good grasp of the details. It also sets the stage for our overhaul of automation as a theory. But before we went there we had to explore what else automation could do for us.

Automation comes in many forms, but the ones most commonly presented are key command logs (copiers), external based automation (cloud or database call connections), and background functionality scripts (prompts followed by data processing). Searching for the right ones to use takes understanding what they are best at.

Copiers are good for tasks that are repetitive in static environments. They work really well if the command is universally executable and all data is present. That is also their weakness: there can be no shift in interface that disrupts it and if any part of the data or source doesn't exist it will fail. It took only one insistent screensaver to push us away from using this for our automation goals.

Connection automation calls up commands and triggers from an external source, often a static one or a default one, much like our reset script did. It isolates automation to counter the weaknesses of copy automation, and has major strengths in reliability of data, maintenance, and repeatability. It is not flawless however, as connections can be poor, no environment that needs alterations is ever truly static, and single point of failure issues.

Then there was prompt automation, which always looks great on the outside. For a user, all you have are a few prompts and results, much like the original Milestones. This makes them very user-friendly and functional. Unfortunately, like many systems that are outwardly calm, the inside is pure chaos, and getting something to run that smoothly means programming for each eventuality or accepting poor function, both of which cost more time.

So it looked like nothing in theory was going to cover everything we needed to automate our tests. Too many variables, too many places it could go wrong, and a system far too large to efficiently program every eventuality. Still, we Dutch are stubborn, so we decided to just dive in and see what the teams could come up with in practice.

6 Automation in Practice

In practice, we needed a process that wasn't GUI reliant, but was also flexible enough to present results in a useful manner. Through experimentation we'd found that automation worked best for us in blocks of functionality, for specific cases and for general integration. But with each team working on different changes, it had to be automation that had maximum customization options without being cumbersome to run and taking forever.

We started playing around with a few test suites focused on customized validations and total control of the test process. These programs worked on the user scale: not coding each action but grouping all functionality into small processes like moving a mouse, searching for items, running general logic and even customized functions. With this we could create GUI, background, and connection automation whenever and wherever we wanted in the process.

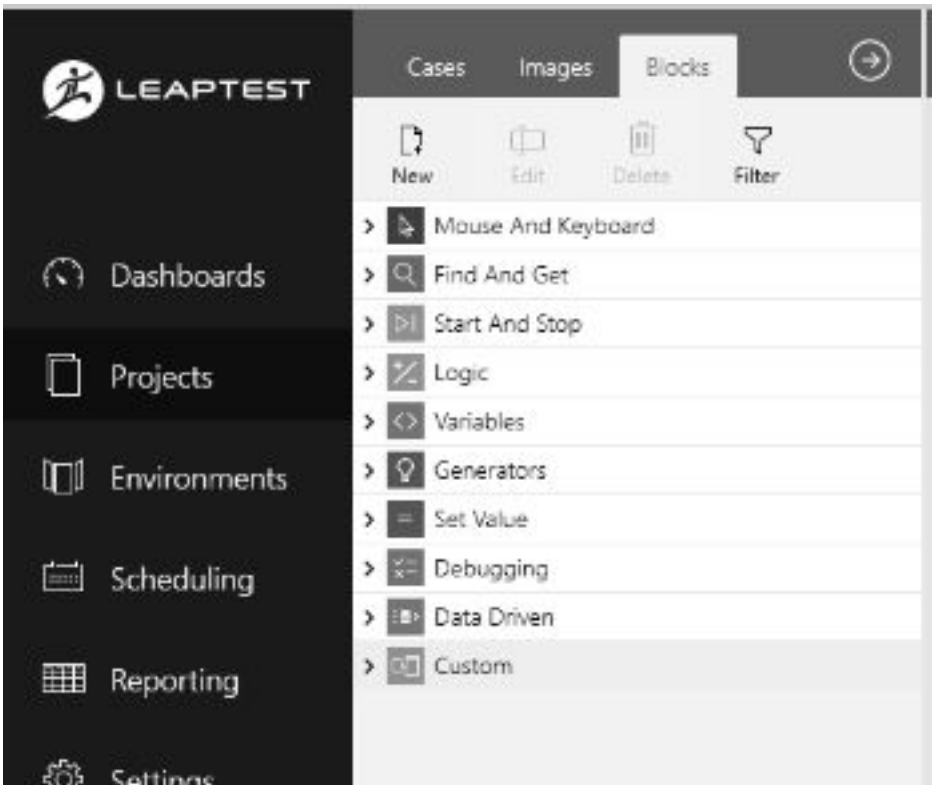

Image 5: Example of a customizable test suite (Leaptest)

This kind of experimentation allowed us to make any automation process our own. We started automating annoying manual procedures like logins, opening programs and starting batches. Small functions wrapped in little blocks allowed us to quickly automate things that just took time, not thought. We originally used it to get the process out of the way so we could automate the testing and verifications, but then the team reflected on what had just been accomplished. Had we just redesigned the purpose of the test suite to automate a process instead of a validation?

7 A New Kind of Automation?

There is a prevailing image of automation as replacing test work. This is a flawed premise, not only in that humans and machines operate differently as shown above, but also misses automation's best feature: companionship. Automation should ***augment*** testing, and by looking at the parts that we didn't like to do anyway, we were able to focus much more of our attention on things we were better at. We created Process Automation.

The team started to focus in earnest on creating shared process steps using the customization functions from suites and from our own systems. We would debate on whether this was a step that:

- Was easily automated
- Didn't require much brain work
- Wouldn't reveal bugs
- Didn't contribute to finding out the quality of the system.

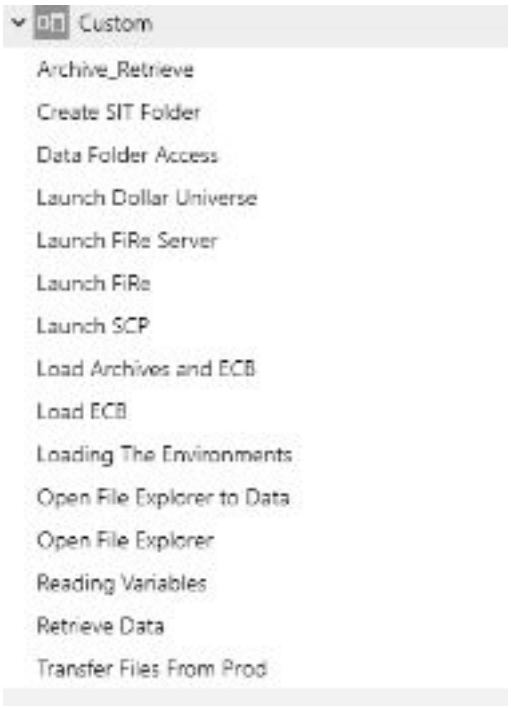

Image 6: Examples of automated processes

If it passed muster, we'd automate it, and before we knew it we had a process that would practically run itself. All a tester would have to do would be to throw the blocks together for the test they needed, and it would run it with an updated setup in an updated environment with the latest data. And once built the first time, the process could be saved for faster recall and repeatability later.

We found that this not only made automation much simpler and enjoyable, it also saved us significantly more time than automating the tests and checks. For an hour's work a tester saved themselves and everyone thereafter at least five minutes of work (or waiting) per test run. That investment paid off quickly in a large system with seven teams.

It wasn't without its pitfalls though, and that is what we are looking at going forward. For one, the process can and does fail, and when it does we don't know what happened or why the more abstract our process automation got. So we started implementing activity logs in any significantly complex automation, as much for maintenance as for debugging. We then sent this log to our DevOps team who used it for monitoring for failures in production, adding more value from our process.

We also learned that automating functionality of a process that large can grow out of hand very quickly. We had to implement functionality in larger blocks, while keeping a repository of the individual tasks kicked off by them for universal maintenance. We're still not sure what to do about the constant growth of functions without in essence recreating the entire system, but it does give us insights into improvements to our existing system by pointing out repetitive functions, combination potential, and sensitive points of possible failure.

Maintenance is often a sore point of automation, so we wanted make sure we learned quickly what the most efficient manners were. Storing values in the code was quickly dismissed in favor of simple excel sheets we could load in with values, and passwords were defined globally so a user wouldn't have to enter it each time it was required in different sections of the process. I'm sure a security tester is having a fit at that last one; if so, we're definitely open for suggestions on how to improve this point!

Automated SIT Template	
Value	Variable
I:\1. Regression Data & Results\	SIT Folder
20170228 1703 March Release	Data Folder
I:\1. Regression Data & Results\Production Files	Data Transfer Site
5	Number of Entities
5201 RXF	Entity 1
utc_cnb	Entity 1 Schema
6016	Entity 2
dubm_drb	Entity 2 Schema
6006	Entity 3
par_cnb	Entity 3 Schema
6005	Entity 4
sch_drb	Entity 4 Schema
5201 FOW	Entity 5
utc_cnb	Entity 5 Schema

Image 7: Excel and Password examples

Lastly, we quickly found that processes are very similar throughout our system. I postulate that that's true in many systems. We found several places where functionality was near enough to another that they could be filled and activated simpler with one function than with several repeated ones. It took us from looking locally to looking globally.

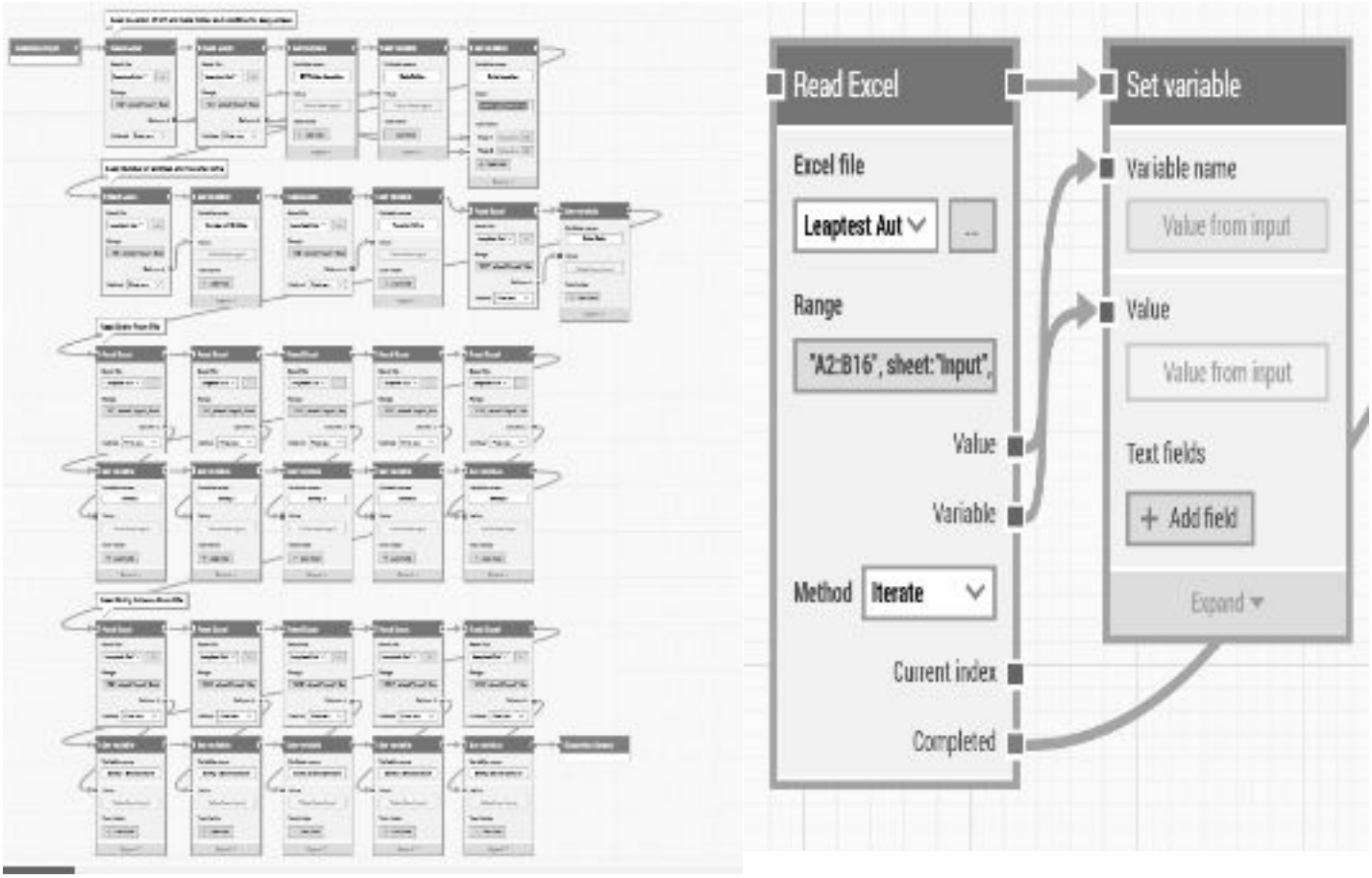

Image 8: Simplification of functionality

8 Conclusion

Automation is a tool. As such, it is up to the user how he or she uses it. We've found that automating tests is a large endeavor with many pitfalls and ignores the selection skills of professionals in favor of massive amounts of data. Because of this we want to push for more process automation, letting technology do what it does best to augment the test process.

In order to do that we want to assist you with the following tips:

- Focus on what you want to produce, and let the automation produce it.
- Look at how you do things now and decide what you want to improve on and what can stay the same (and automated!)
- Think about your approach; learn from others and from new technology before diving in. Learn especially about the strengths and weaknesses of your team and your tech.
- That said, don't be afraid to experiment to find things no one has before or new applications for old approaches.

Lastly, don't hoard automation. Automate the process for others as well as for yourself. This helps in recycling of tests, saves others time in designing how to monitor the process, and even gives your users a break by granting them an easy to use tool for User Acceptance Tests.

By doing this our hope is that more people can build the automation necessary to make their testing and overall system quality continuously better and set out faster On The Road To Quality!