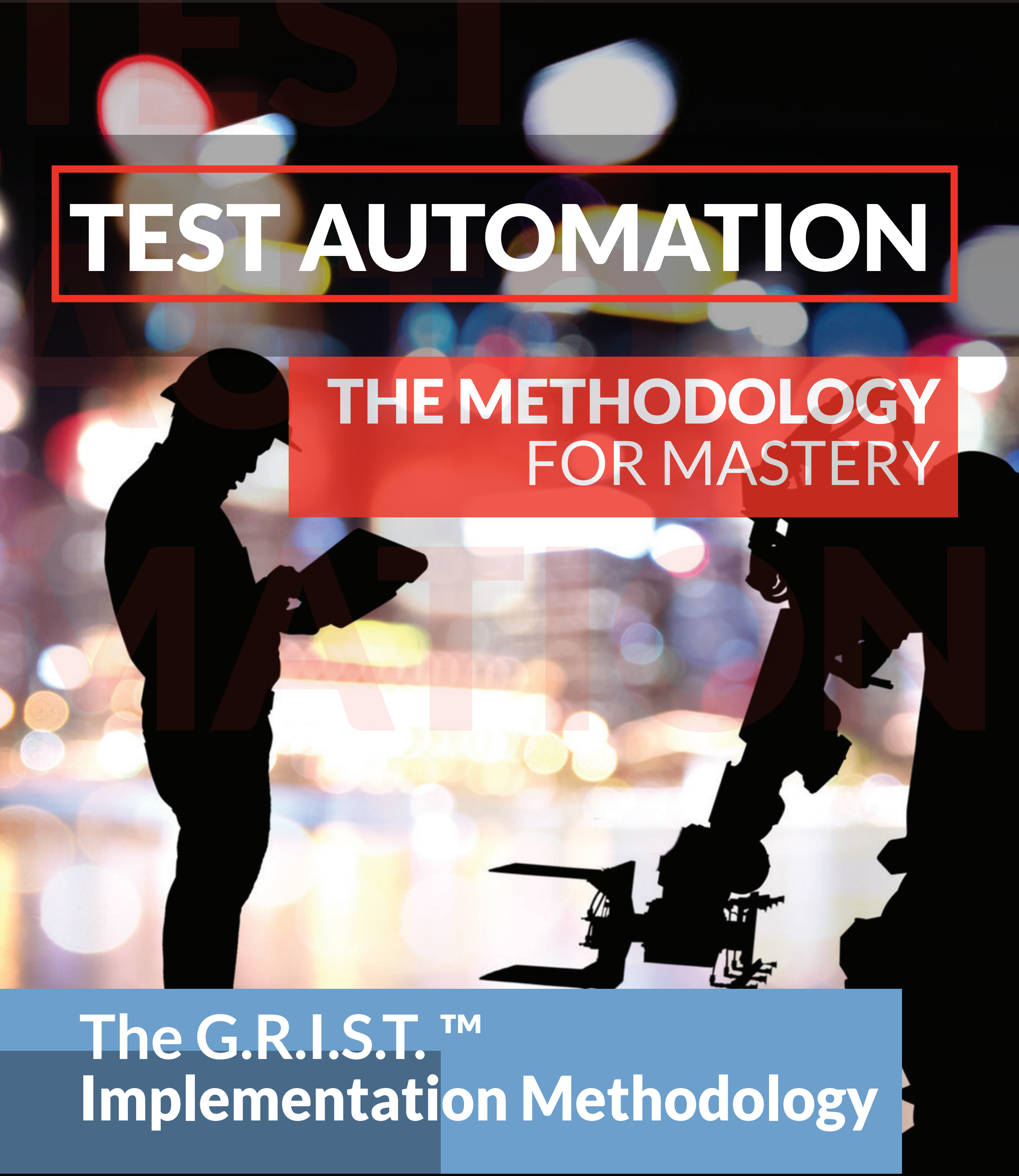
# TEST AUTOMATION

## THE METHODOLOGY FOR MASTERY



The G.R.I.S.T. ™ Implementation Methodology

AUTOMATED-TESTING.COM

"Of the $80 billion spent annually on testing just 3% is spent on test automation."

# INTRODUCTION

A startling statistic. Why such a small proportion? Possibly because too many automation projects have failed over the years. Possibly because still too many test automation projects are doomed to failure (even today!). The tough reality is that a large number of automation projects never make it past the first few months.

Many issues contribute to this high failure rate. Yet many of the issues can be avoided with a clear approach. This book details a 5 point approach that helps to take you through those "make or break" first few months.

The majority of failure are not down to the tools or the people using them. The tools and the people are good. Good people. Good tools.

It's the approach taken that's flawed.

For many there's an attitude of let's "buy the tool" then just "get some training". From there they expect their team will just start to build automated tests instead of running them manually. Get the right tool, get a bit of training and we'll be fine! Err... no.

It's an approach that is flawed. A flawed approach that fails most of the time! You can have good people and good tools. That counts for nothing if you have a flawed implementation approach.

A flawed approach that leads to failure.

Failure because teams put absolutely no thought into how test automation is going to become a way of life for them. They put no effort into changing their systems, processes or way of working.

Test automation is a complete mind set change for a team.

It requires a completely different way of approaching testing. Test automation is more a DEVELOPMENT PROJECT than it is a test project. Automation is coding after all.

**TOOL AGNOSTIC**

**IMPLEMENTATION MODEL**

Implementation success comes with applying a proven approach. We've taken the term "GRIST" to help you focus on what's important. G.R.I.S.T. is an implementation model designed to help you see where you need to take action to deliver the test automation results you demand.

**G.R.I.S.T.** is an approach that focuses on structuring and overseeing the implementation of automation in the Agile development process. It's more top down than bottom up. More "think through the system" for ensuring agile team engagement up front. Less "start building things from scratch". More about picking the right tactics to employ once you know what it is you really need to deliver. In short – a little more thought up front and far less wasted effort at the back.

## 1. GOAL, STRATEGY & OBJECTIVES

Define what you want to achieve. Once you know what you want, you can work out how you're going to get there and which route you need to take.

## 2. REPORTING

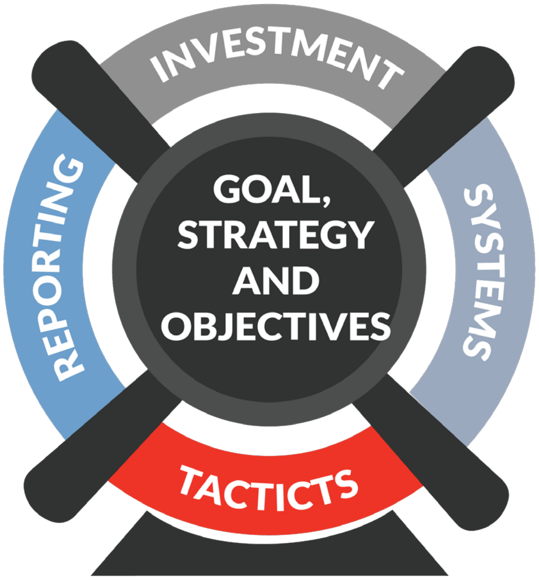
Track your progress towards your goal. You might be surprised to find out that we're not talking about the usual test reporting components here! Tracking implementation progress is critical.

## 3. INVESTMENT

Test automation is expensive. Whether it's tools or people, the outlay is significant. Make sure you're tracking and seeing a return on your investment.

## 4. SYSTEMS

The processes and systems are important. Putting things in place at the start save you time and effort in the long run. Get this right and larger organisational roll outs become far simpler.



## 5. TACTICS

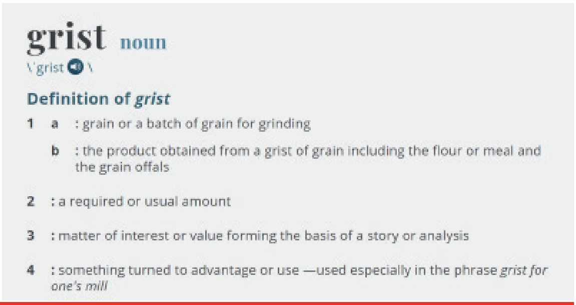
The nuts and bolts. How you construct and implement the technical aspects of your chosen solution that's designed to meet your objectives.

What you need is an approach to implementing test automation that works. For that we've come up with the little acronym …

## "GRIST"

This little word is the basis of a methodology that helps you ensure that test automation becomes a way of life. It's the basis of a little check list to make sure you stay on track.

The word GRIST meaning...



**grist** noun

\ˈgrist \

**Definition of *grist***

1 a : grain or a batch of grain for grinding

b : the product obtained from a grist of grain including the flour or meal and the grain offals

2 : a required or usual amount

3 : matter of interest or value forming the basis of a story or analysis

4 : something turned to advantage or use —used especially in the phrase *grist for one's mill*

**G OALS, OBJECTS AND STRATEGY***

**R EPORTING**

**I NVESTMENT (RETURN ON)**

**S YSTEMS**

**T ACTICS**

It's the definition "something turned to advantage" that is most appropriate. That's not important though. What's important is what these letters stand for.

** - you've probably already completed this step. Possibly all in your head, rather than on paper. Either way it's worth having a bit of clarity on these points before you start.*

In the next few sections we're going to take you through the G.R.I.S.T. methodology explaining the concepts and approach that will hep you succeed with automated testing.
A methodology that will take you away from this "purchase a tool with a bit of training" approach. A methodology that will have you thinking of automation as a change to your way of working and way of life.
A change that starts with having your Goals clear right from the start.

## SECTION 1 : ARE YOU BUILDING A LADDER OR INSTALLING A LIFT?

**“If the ladder is not leaning against the right wall, every step we take just gets us to the wrong place faster.”**

– *Stephen Covey*

I'm pretty sure Stephen Covey didn't have Test Automation in mind when he wrote this. Yet it's very apt. Too many teams lean their automation ladder up against the wrong wall. They aim for the wrong goal.

No prizes for guessing what the 'G' in GRIST stands for then! Well actually it's about more than just the goal. It's about...

## GOALS, STRATEGY AND OBJECTIVES

Why should you care about this? Because most companies spend a huge amount of money and time building test automation and don't even realise that they've built the wrong thing!

It's not uncommon to see a solution that is:

**A. so complicated and un-maintainable that only a few automated tests run reliably.**

**B. Nine out of 10 people in their team find it too complicated to use.**

**C. The rate at which reliable automated tests are added is painfully slow.**

You only need take one step back too see that it would have been cheaper, and better, to have just run the tests manually.

This is why you need to be clear about your Goals, Objectives and Strategy up front. Then.... and only then.... will you build the RIGHT solution for your team and your project!

To borrow from a good article on Medium these aspects can be defined as follows:

GOAL—A broad primary outcome.

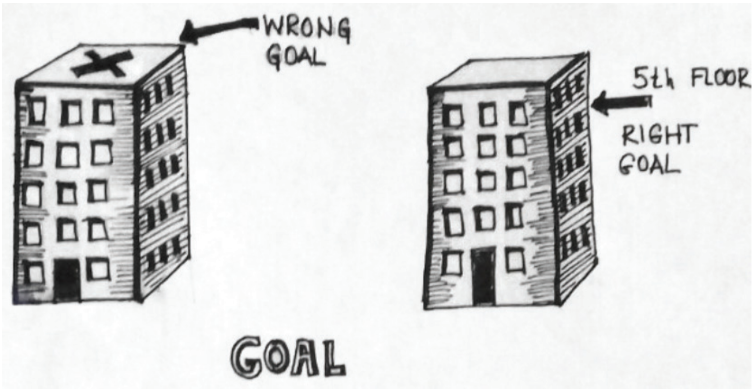
STRATEGY—The approach you take to achieve a goal.

OBJECTIVE—A measurable step you take to achieve a strategy.

TACTIC—A tool you use in pursuing an objective aligned with your strategy.

Now Tactics we'll come on to in stage 5 of our GRIST methodology (no - no prize for guessing what the 'T' in GRIST stands for either). Most inexperienced automation engineers start with tactics and build up from there. They find a Selenium framework, pick a coding language and start building. And by that time it's all toooooo late!

Anyway, let's go back to our "Ladder" analogy and look at Goals first.

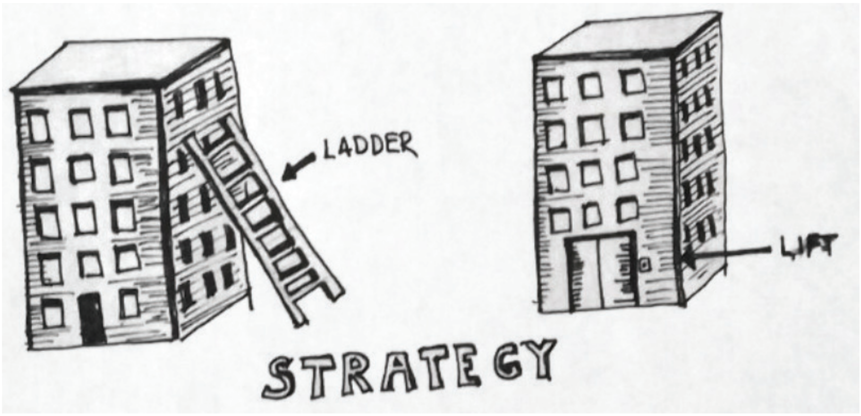


Think carefully about **YOUR GOAL**. Do you really want to invest significant amounts of time and money to be right at the top? Or would the 5th floor be a more reasonable goal?

Your goal might be, for example:

'Within 12 months have a regression pack containing all our high priority +ve test scenarios for product XYZ. This will run every night in our stable test environment. Everyone in our team should be able to contribute new tests and maintain this regression pack.'

You should be able to asses your success against this stated goal. It needs to be attainable. For example there is NO point writing automated tests if you DON'T have a stable test environment to run them in!



If you need to get to the 5th floor, is a ladder really going to cut it? Or should you be looking at a strategy based on installing a lift?

You can go designing and building your own ladder if you want! BUT, you'd probably be better off paying someone to install a lift... Expensive? Yes. Easy for everyone to use and faster? Definitely, YES!

A well defined goal will help you work out strategies that you can apply work here. If you want everyone in your team adding automated tests then don't build something that requires a lot of coding and a good understanding of OOP.

With your goal in mind then, answering these questions will help you identify **YOUR STRATEGY:**

- Are you going to bring in a full time automation engineer?
- Are you going to use existing development resources or outsource?
- How are you going to find time to write automated tests?
- Will you have budget to purchase tools that allow you to implement faster?
- Will you use open source tools that will suck up development resources?
- Roughly what will your approach to reporting and tracking be?
- How and where will you define your processes and systems?
- What will your approach be to building team engagement?

If your goal is to have everyone in your team writing automated tests you might start out defining a strategy along these lines...

Strategy Option 1:

'We'll purchase an easy to use automation tool, build a new dedicated stable test environment, invest in setting up a continuous test process......'

Another strategy you could use might be along the lines of....

Strategy Option 2:

'We'll out-source our automated testing, providing detailed test case definitions to an off site company who will......'

Both valid strategies but only one will get you to the goal of having everyone in your team adding and maintaining automated tests.

Your goal should lead you towards the type of strategy you'll apply.

And your strategy leads you neatly into your Objectives....

Think of **YOUR OBJECTIVES** as a few targets that will get you to your goal, using the strategy you've settled on.

If you need to get one person to the 5th Floor, once every 3 months then a long (very long) ladder will do. If you need to get a lot of people to and from the 5th Floor 24 hours a day then building your own ladder isn't a smart move.

Start to map out a path towards hitting your goal. For example...We'll reach point X in 1 months time, hit target Y within 3 months and arrive at our goal within 12 months. An objective might look like this then,

"Our objective is to write 10 automated tests a day and execute every test in our automated regression pack at least once a week."

Success with automation is largely about adding good numbers of, high quality, reliable tests over a decent period of time. That's why having good objectives is important! **Good objectives will also help you see if your strategy is appropriate!!** If you start building your own ladder your chances of hitting that objective are slim!

At this point we've defined our goal, come up with a strategy that we like and we've set some objectives. **Now we need to iterate.**

Work down again from the goal, through your selected strategy and to your objectives. Then consider them all in relation to each other.

***If your goal is to get lots of people to and from the 5th floor of an office block, AND you need to do that every hour AND you have the funds to install a lift .... THEN everything fits together.***

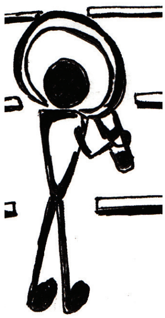
***If you're looking to get to that 5th floor AND you can only afford to build your own ladder AND the people you have to do the job are scared of heights.... THEN you might as well give up now.***

I'd prefer it if you didn't give up now though. So long as your Goal, Strategy and Objectives all fit together there's no reason why this shouldn't be a resounding success.

That is, so long as you've thought through the 'R', 'I', 'S' and 'T' in GRIST too.

## SECTION 2 : WHAT'S REALLY IMPORTANT?

**"If you can't measure it, you can't improve it."** – *Peter Drucker*



We're on to the 'R' in G.R.I.S.T. and considering our reporting requirments. The usual pass/fail test results are important BUT they're not really important.
In fact they're absolutely worthless if they're wrong or not showing you what matters.
What is important, is data that shows you how YOU are doing. Early on you want to focus on the quality of YOUR automation system. Not the quality of the application under test.
I can not emphasise this enough. This is one of the most important success factors. At the start of your automation project track data that shows you how your automation project is progressing.

Why? Because in the early days 'shiny object syndrome' will be clouding everyone's view. It's not that carefully crafted project plan. It's not that new automation tool you invested thousands in. And it's not that external consultant you brought in. It's NOT those things that ultimately make automation a success.

**What makes it a success is .................... a day in, day out, HABIT.**

**“Motivation is what gets you started. Habit is what keeps you going.”**

*– Jim Rohn*

“Fantastic!” You have the budget and resources to employ a full time team of automation engineers (by the way - if you can afford it this is the best way to deliver on test automation). It's just that many of us don't have access to dedicated automation resources. We don't have a dedicated automation engineer in our compact little Agile team.

Keeping the momentum going on your automation project is probably going to come down to you and a couple of others in your team. All of whom have a million and one other tasks to do.

So my question to you is this....

“How will you keep going with test automation on days when every-thing else feels like it's a higher priority?”

In my experience it comes down to habit. You'll want to develop your automation efforts into a habit as quickly as possible. One way to go about this is with an approach known as ....

“Using accountability to stick to your habit”

In short though, using accountability means demonstrating to your peers that you’re making progress.

Think about finding a way to measure and report on a few small aspects of your plan. I’d suggest tracking on three things....

1. **Test Development**
2. **Test Environment Stability**
3. **Test Case Quality**

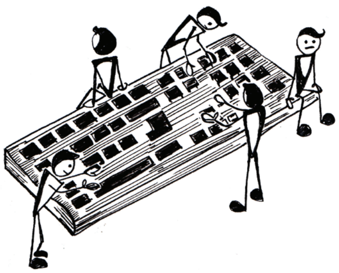
From the outset you need to start thinking about how you can improve your processes and systems around test automation. To improve these processes you need to measure them. And you need to make those measurments visible!

You should assume that at some point things will start slowing up. If there’s no visibility of this you can guarantee that eventually things will grind to a halt. If there is visibility then you’ll know when to start taking corrective action.

If you’ve set some objectives (See the ‘G’ in G.R.I.S.T.) then you can monitor your progress against those objectives. You don’t have to be totally accurate. It’s not the exact numbers you care about. What you care about is the trend!

You want to see data and trends on these three things:

## 1. TEST DEVELOPMENT



No point having the most advanced framework and infrastructure if you're running only a handful of tests that give you very little coverage. You want to get into that habit of adding tests every day. You might start developing small, easy to automate tests. Just make sure you start and that you get into that habit.

A simple chart showing number of automated tests added is a good place to start. It's usually quite simple to automate the creation of this chart too. However you do it, you and your team, need visibility of how many new test are being added each day.  This is a simple but effective way to keep you on track.

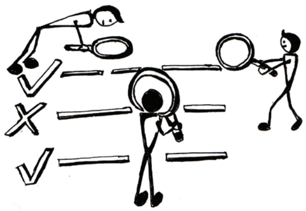
## 2. TEST ENVIRONMENT AND TEST DATA ISSUES

Getting to the point where you and your team have complete confidence in your automated test results takes commitment. You will be challenged by test environments that don't behave as expected. You will struggle with test data that's set correctly on one test run and invalid on the next.

You'll want to track when tests fail due to environment or data issues. Maybe you're better off focusing on improving test environment stability rather than writing new tests? Reporting in this area will help convince others that something needs to be addressed here.

## 3. TEST CASE QUALITY

In the early days of implementing test automation your tests will be unreliable. It's code testing code. Your automated tests are going to need 'testing'. Picking up on unreliable tests early on so that they can be fixed or eliminated is essential.
You do NOT want to spend a lot of time analysing failed tests 'where the issue is' the "test" and not the "application under test". Focus effort on getting tests stable right up front. It'll save you significant amounts of effort later.



In the early days take it as a given that your test environment will be unstable. Your tests will be unreliable. That you'll be distracted by a million other competing priorities. If you think you'll end up with a perfect automation process from day one .....you're in for a shock.

If you have a few simple data points to identify issues early you'll be able to correct things early.

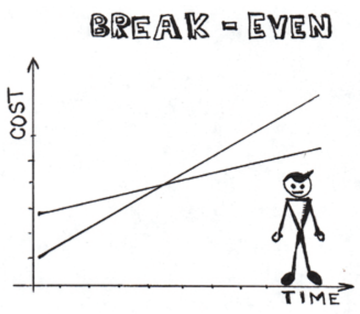
***Make sure you have a few 'publicly visible' data points that encourage you into the "habit" of writing good automated tests.***

## SECTION 3 : ARE YOU REALLY SEEING A RETURN ON YOUR INVESTMENT?

Ask yourself this....
"Which option is going to be more cost effective, more reliable and faster? Manual or automated testing?"

This is a difficult question to answer. Many companies waste a lot of time, money and effort finding out the hard way.

Developing stable and reliable automated tests in complex environments is NOT easy. It's not always a quick win. It's a significant investment.



The 'I' in G.R.I.S.T. Stands for "Investment". To be more accurate it's the "Return on Investment"

Easy to calculate when viewed like that. Not so easy in the real world. What's worse is that it tends to imply that it's a one off calculation, defined at the start of the project. It's a figure that's just plugged into some business case from day one.

At the start this calculation is usually nothing short of a COMPLETE GUESS. A guess that's NEVER validated. A guess that's never adjusted.

**What if....** you actually tracked this over time?

**What if.....**You actually put real figures in each time you ran a test?

**What if...**. you could visually see when you're going to break even and start seeing a return?

**What if....**. you could spot changes in your rate of return so that you could take corrective action?

This is where we want to get to. If you have this information then you'll be able to answer three critical questions...

1. Should we continue to pursue this?
2. Are we happy to continue making a loss whilst we work towards making a return?
3. What do we need to adjust to make sure we ARE making a Return on our Investment?

Trouble is you can't answer ANY of this until you work out what your Return on Investment really is. And, to get to that point you need to understand…..

***The Issues with Traditional RoI Calculations***
Many automation projects don't provide an RoI. Harsh fact of life! The question is how do you know? Well most teams don't know because they don't pay attention to three key issues.

**First** IT'S NOT a static value. It is not a "calculate once" then forget exercise. It is not a guessed value that's just fed into the business plan to get justification for your automation project. RoI is something that should be calculate regularly. Even dynamically. Calculated every time you create and run tests.

**Secondly**, most companies miss out figures like test maintenance and test environment management. Both of these suck up significant amounts of time. This all eats into into your RoI. Start tracking this and you can start fixing it.

**Thirdly**, the RoI data is not detailed enough to help you improve. Maybe it's just a handful of bad tests that result in your poor RoI. Maybe it's a poor RoI on the tests written for a complete application. Maybe the application doesn't lend itself well to automation. Maybe you've bought the wrong tool. Maybe the automation engineers don't have enough experience. Either way you need to find a way to identify these issues.

YOU don't know any of this until you start tracking the RoI.

So what does happen when you ...

A. Track RoI continuously?
B. Track all the data points that are important?
C. Have figures that help identify where the issues are?

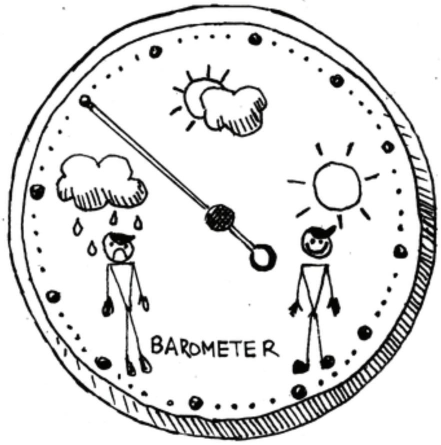
When you start tracking and identify where the issues are, you learn and you improve.

In short you get an early warning system that directs your actions!

***The Test Automation Barometer***

Why do you really want to track RoI in test automation? Because when you track this right you get an effective early warning system. An early warning system that'll help you see bad weather when it's on the way.

We get so caught up in the day-to-day process of 'doing' automation. We forget to look up and asses what it's there for. When you have the RoI barometer in place you have that daily weather forecast. A reminder that you need to poke your head out of the window and see what's really going on.



Don't get me wrong. It's NOT the ONLY indicator. This has to be considered in conjunction with other indicators. You may accept a poor RoI in the first few sprints as skills build and the volume of tests increases. What's important though, is that YOU KNOW!

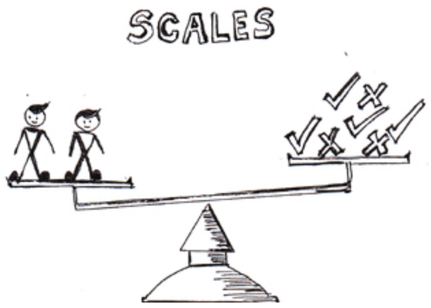
Maybe it's just fine that you don't see a return on the investment for 6 months. It may be that automation gives you something you can't very well do with human testers. You may accept that this is an investment for the future. An investment that comes to fruition as the process matures throughout your organisation.

You'll never know though without a barometer. With a barometer you get see when things start changing. With a barometer you know!

And when you know, you can adjust. You can tweak. You can improve.

***Tip the Automation Scales In Your Favour***

With a barometer in places it far easier to get in to the 'looking for improvements' mindset. When you're in that mindset it's far easier to tip the test automation scales in your favour.



You'll find yourself looking for the easy wins upfront. You'll start finding the simpler routes to automating tests. You'll have RoI at the front of your mind and be focused on getting a fast pay back from test automation.

Ultimately this is about finding the most efficient, cost effective, way to improve the quality of your releases. If you just think implementing automation is the solution then sure, you might want to start out with automation. You just wont see any real benefit.

Understand that automation is expensive but worthwhile when you're focused. Having a barometer in place keeps you focused.

That barometer allows you to demonstrate - with evidence - that the investment is paying off.

## SECTION 4 : WHY IS IT ALL ABOUT THE SYSTEMS?

**“There’s a huge difference between being a replaceable cog on the assembly line and being the one who is missed, the one with a unique contribution, the one who made a difference.”**

*– Seth Godin*

How can you make that transition from replaceable cog to the one delivering the unique contribution? Step back and focus on the test process ‘Systems’.

And guess what? The ‘S’ in G.R.I.S.T. stands for ‘Systems’. We can start delivering that unique contribution when we start focusing on constructing better systems rather that being a small part in existing systems.

It’s the design, construction and improvement of ‘Systems’ that’s key to test process improvement. Systems to take care of common, day to day, test processes. We should be building systems and processes that make everything repeatable and automated. Not automated in the sense of automated tests. Automated in the sense of test processes operating on autopilot.
This is the concept of ‘systatmising’ (I really dislike this word but if

feels appropriate here) the processes you follow. You take a test process that your team follows. Then you put a system in place that runs that process like a production line or conveyor belt. Then you have repeatability. With repeatability comes consistency, reliability and speed.

## What You Don’t Want Is Inconsistency and Unreliability

Inconsistency and unreliability creep in when you DON’T pay attention to building the systems. If you don’t look to build and improve these systems everything slowly comes down to run at a snail’s pace. Ultimately you end up with test processes that you have no confidence in. You end up with something that doesn’t meet the demands of today’s fast paced agile teams. YOU fail to deliver.

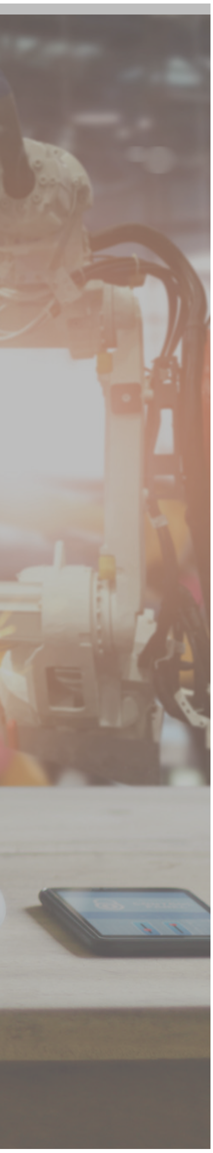
However, get the systems right and you’ll be able to concentrate on creating the quality and quantity of automated tests that will make a difference. Less time wasted on day to day trivia and more time focusing on what matters. What matters is designing, building and running great tests.

## The Three Most Important Systems

The three most important systems from the automated testing perspective are....

1. **Test Development and Test Execution**
2. **Test Environment Management**
3. **Test Data Management**

Let’s take a look at each of these in turn.



**1. TEST DEVELOPMENT AND EXECUTION**
Look to streamline the development, testing and execution of your automated tests. Most of this process can be put on autopilot. Granted the test automation development step is somewhat of a manual process (we haven't quite hit the point where AI is designing our tests for us). The rest (source code control, testing the automated tests, executing the tests, etc) should be on autopilot.

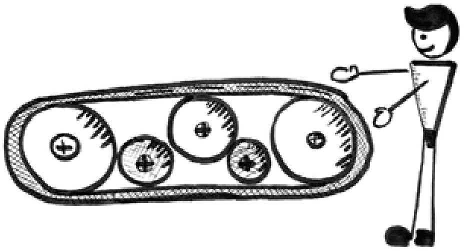
**2. TEST ENVIRONMENT MANAGEMENT**
Test environment management is never an easy one. Most systems we're testing have tentacles reaching into so many other systems. That makes it difficult to reliably instantiate new instances of your test environment as part of an automated process. Difficult but well worth investing time in. Clean, well managed, test environments simplify so many other parts of the systems we build. The payback for the effort invested in this area is significant. Don't shirk this area because it looks too difficult.

**3. TEST DATA MANAGEMENT**
Test data management is another difficult, but essential, area. With start up projects it can be easy to backup and restore databases for test environments. Trouble is in start up mode most teams don't think this is a priority. Then it gets left till it's not so easy to address. And it's not so easy once the system has grown beyond it's embryonic stages. Sometimes just having scripts to monitor your test data can be a simpler solution. An alert that tells you your test data has been compromised is usually a good first step. Reliable automated testing depends on you solving this problem.

All of these parts are essential parts of a CI, CD and CT approach. The parts you focus on will probably be part of the bigger picture. What is essential though is that you need to solve these issues. You must get to a point where you have a reliable automated testing capability. You need to solve these three problems with systems and process that you can operate on autopilot and depend on.

**Build Your Conveyor Belt**
The solutions you develop around these three core systems are critical. Critical because the repeatability these systems deliver give you conveyor belt like….

A. Consistency
B. Reliability
C. Speed



Consistency means that everyone in your team knows what's happening. Reliability helps build trust across the whole team. And speed... well who doesn't need higher release speed these days?

This should all be running like a conveyor belt. You start with test design at one end of the conveyor belt. Then automated tests drop off the other end of the conveyor belt. They drop off as reliable automated test that everyone can depend on.

What you do NOT want is people in your team manually walking bits from one part of the factory to the next. Manually tweaking and fiddling with different bits and causing bottlenecks. A well designed conveyor belt avoids this. Well designed 'Systems' avoid this.

More than this, well designed 'Systems' free up time to let your team do what they do best. What they do best is designing and writing good tests.

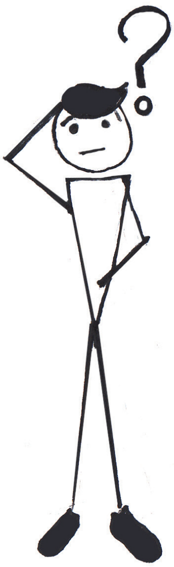
## SECTION 5 : IT ALL COMES DOWN TO TACTICS

**“Strategy requires thought, tactics require observation”**

*– Max Euwe*

The ‘T’ in G.R.I.S.T. is for tactics. The last component of the G.R.I.S.T. Methodology. As Max Euwe says this is more about the observation and the actions you take on a day-to-day basis.

If you’re wondering how your Goals, Strategy and Tactics all relate then there’s a great definition of the word ‘Tactics’ in the Business Dictionary. Here they define tactics as a...



*Means by which a strategy is carried out; planned and ad hoc activities meant to deal with the demands of the moment, and to move from one milestone to another in pursuit of the overall goal(s).*

We defined the goals and strategy when we looked at the ‘G’ in G.R.I.S.T. Now we’re on the final component. The Tactics that will drive us towards the goals we set at the start.

The only slight problem with tactics is that it’s all encompassing. It covers a multitude of things in our automated testing game.

Everything from the ability to write code, skills to build systems, the creativity to design tests, the due diligence to analyse test results regularly, etc.

You need to decide which tactics to use in each of these areas. You need to decide after observing the situation and considering your overall Goal and Strategy.

**Which Tactics Should I Use?**

Which tactics you use will come down to the types of people you have in your team. It will take into account the tools you have at your disposal. You'll also need to anticipate the demands of the application you're testing.

It's very difficult to be specific about which tactics you should employ in your specific project. Every team, every application under test and every tool set demands different tactics. And of course depending on what stage you're at in a project will dictate which tactics you employ at that point in time.

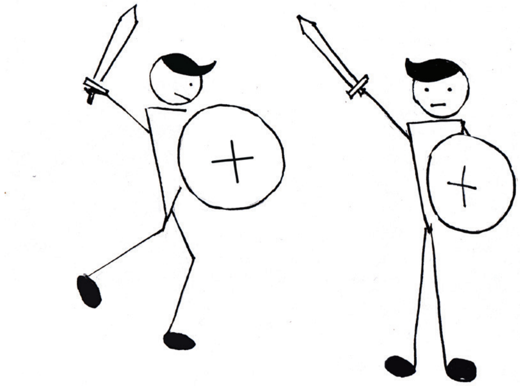
*You have to pick and deploy the right tactics to mould your team into the shape that is best suited to delivering your Goals, Strategy and Objectives.*

Let's take a simple example to start with. Let's say your objective is to develop automated tests within your team (not outsource to an external team). Your tactics would revolve around learning about automation tools and picking up test development skills (coding).

Alternatively, If your objective was to outsource your automated testing then your tactics would revolve around things like learning to manage remote teams, defining clear test specifications and agreeing delivery criteria. Your tactics would be more about the 'managing' rather than the 'doing'.

## TYPES OF TACTICS

There are many ways to look at tactics. Hundreds, possibly thousands, of different tactics you could employ. You can add to the complexity here by defining many different ways of categorising tactics too.



To keep it simple lets just spilt tactics down into 2 general categories:

1. **DEFENSIVE** - reactive actions that deal with what's going on right now

2. **OFFENSIVE** - proactive actions that anticipate what you think will happen in the future.

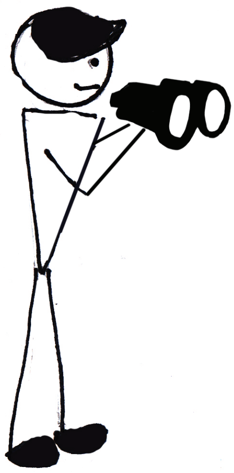
Maybe you're thinking about which action to take to solve a problem or improve part of your process. Rather than just pick the first tactic that comes to mind think about your overall goal and objective. Then you might think "do we need a Defensive or Offensive tactic to help us hit that objective"?

## An Example

A couple of examples might help. ***Your automated tests are failing because of stability issues.*** You need to employ tactics around building robustness into your automation solution. You can employ defensive or offensive tactics to solve this. You could even employ both. ***Defensive*** - Short term deploy code fixes targeted as specific automated tests that have problems. Turn off automated tests that aren't reliable, can't be fixed easily and waste lots of analysis time.

***Offensive*** - Medium term you might decide to look at implementing a new framework or even see if you can find a new tool. You might convert some GUI tests in to API tests. You could look at solutions that makes it easier to develop stable automated tests in the first place.

This way of thinking your actions through helps because it forces you to take a step back. You take a step back and think about the best tactics to deploy to hit your overall goal.



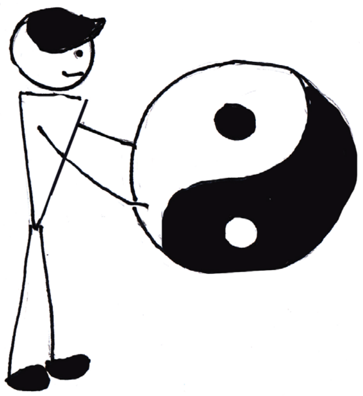
Yes you might need to do something right now to solve a problem (Defensive). Is this initial tactic going to take you towards your overall goal though? If not what's the Offensive tactic you need consider too?

**The Yin and the Yang**
Keep the following in mind when you think about tactics. Remember that it’s more about the actions you take when responding to current events and conditions. Day to day demands may force those tactics in a particular direction. They may even drive you away from your stated goals and objectives.To keep things in alignment it’s worth looping back from the ‘T’ in G.R.I.S.T. to the ‘G’.  Make sure your Goals are still relevant. Spend some time making sure your Goals and Tactics are aligned.

Are the tactics you’re employing day-to-day really aligning with your Goals? If not maybe it’s time to look at re-evaluating your Goals and the associated Strategy. No shame in that. In fact a recalibration of the Goals and Strategy is one of the healthiest things you can do.

Or maybe the tactics you’re employing aren’t the ones you need in order to hit the goal. In which case maybe it’s time to think harder about the tactics you implement each day. Maybe fundamental changes in your team and/or technology are required to enable you to employ the tactics you really need.
Either way the success of your automation projects depends on clarity and consistency. It depends on you making sure everyone is pulling in the same direction. Everyone looking to achieve the same goal. Employing the right tactics to hit that goal.

Often the tactics you feel you have to employ may seemingly be at odds with the goals you want to hit. Your tactics might seem contrary to your goals. Yet these seemingly contrary forces can be more com-



plimentary than you thought. There will always be a bit of "Yin and Yang" between your Goals and your Tactics.

**This is healthy. You may as well embrace it.**