Shift Left Group
Making Business and Technology Better

Automation Smells

Top Level Automation Improvements for Non-Automation Experts

By Richard Edwards – Head of Automation at Shift Left Group

Why you need this article in your life

As with many things in the world, you often can smell a problem before you find it and functional test automation is no exception. The trick is recognising the difference in warning between the rancorous stench of rotting meat and the tantalising aroma of excellent cheese.

The purpose of this article is to help you identify bad automation smells in your delivery lifecycle without the need for a nose that has been honed from years of technical automation work.

After 15 years in technical test delivery, working with dozens of clients and even more projects; I've compiled a list of smells that I seek out when supporting clients which can highlight low levels of adherence to automation best practice. By either asking direct questions or keeping a nose out for suspicious odours you can quickly identify any automation inefficiencies or impediments and recognise the need for improvement. These automation smells apply to both classic waterfall and modern Agile delivery methodologies. The content and supporting stories are not theory, they are summaries of my first-hand experiences.

The overall goal is to stimulate continuous improvement and unleash the full potential of automated testing. We do not want to create a deodorant to mask odours but rather identify and remove the source of the nauseating hum.

Why you need good automation in your life

Automation is an investment in time, money and resources and is key to supporting an efficient software delivery process.

While the end product is handled by everyone, it's typical that the automation is mostly managed by the Quality Assurance team as few others have the required skillset or are empowered to develop it. I would say we're all responsible for quality and don't let automation best practice become a blind spot in your IT project lifecycle.

It stinks that we have no idea what our automation has actually tested

The notable odours:

- Key people have no idea what runs in the automated pack
- People feel automation is running the wrong tests

Question: What are you testing?

It's easy to get carried away and forget that automation is for regression tests. We don't need to automate all tests and we don't need to exhaustively test the application for each release. Especially for automation at the GUI level, fewer tests mean less maintenance and less technical weight carried forward in each release. A good regression pack is the right size to mitigate the risk of regression issues. Someone is paying for the automation to happen, so, it's important for that someone to get a positive feeling that automation is delivering value.

To know what you're testing, detailed traceability helps. All tests should have a test basis and knowing that information source helps translate the technical delivery to the business need.

In this situation a relatively easy approach is to have a good set of reports and to seek feedback from the stakeholders. Sharing what you've automated as you go and a future plan (if you have one) helps prevent conflicts down the line. Outline the rules for selecting regression tests and allow the team to share the task of identify regression candidates.

Story: I did an automation review for a client. An external third party was doing the development and all test testing - including GUI automation. I was asked to get involved because the management didn't trust the automation. They felt it was a waste of time and effort, as there were frequently uncaught regression defects creeping in across releases. Upon a detailed review one of the problems was transparency as reporting and tracking of automated assets was almost non-existent. The third party managed it all internally and the client management team had no visibility of what had been automated or what the future steps were.

Moral: Clear reporting was a core recommendation of the automation review. The aim is to keep detailed traceability of tests and be as transparent as possible. When you know what tests you will automate and why, share the list and allow input and adjustments – this is more than just a test activity. It's for the client, the BAs, the business SMEs, the developers and anyone else involved as they will all have an understanding of what is critical and which parts of the application are likely to be risky.

It stinks that our automation takes forever to run

The notable odours:

- Automation takes a long time to run - too long for the delivery methodology
- The automation pack has far too many tests
- There are automation execution bottlenecks

Question: Why does automation take so long to run?

One of automation's great benefits is fast feedback – and fast is a relative term. Days isn't fast but it can be acceptable especially if it's still faster than manual testing and doesn't have the reliance on manual resource. The real reek here is if automation is too slow for the delivery approach.

Test teams and automation are often left with a "do-what-you-can" situation to get tests running which often results in non-optimum solutions. Automation is an investment and it's logical to keep that investment going and not stop halfway through. As the automation return on investment is delivered this creates its own quantifiable business case for more automation and more support for automation requirements.

Story 1: I was visiting a client and someone mentioned the automated test pack of about fifty scripts was taking 3 days to run. A little digging later and the challenge they faced was they needed overnight batch jobs to mature the test data between the days. The client believed the tests were only valid if the overnight batch job executed on its own.
The core recommendation was to separate the purpose of the tests. Initially, there was a bit of education that the overnight batch job is just a batch job on a timer. It can be kicked off at any time to progress the system data with the same functional result and automating this would bring down the execution time to a few hours instead of the 3 days. The testers (and automation) would just need access to the system and server which managed the job. Following that, if the batch job's execution schedule was classed as critical regression coverage, it has value creating new tests specifically for it. This provides significantly more accuracy and result clarity than waiting overnight for it to run and seeing what has happened the next day.

Moral: Don't stop the investment in the automation. Just because there is an impediment that prevents automation running – remove it or create a way around it. Support the QA team in managing aspects of the integrated test environment to provide the greatest efficiency.

Story 2: I was part of a delivery team for a client. The automation took approximately 8 hours to execute.
In principle this was fast enough - it was capable of overnight execution. However, with an adjustment to the source test data and little upgrade, it was possible to parallelise the tests and bring the run time down to 2 hours, which enabled within-working-day feedback. In this instance, it was relatively easy as it was an open source solution - however, even if it was licensed software there would have been positive cost-benefit result for buying more automation test tool licenses.

Moral: Don't feel limited by what you have. If there are speed improvements vs cost - create a business case for it. Overnight automation is great, but a two hour feedback loop is better.

It stinks that maintenance of our automation takes so long and costs so much

The notable odours:

- Automation rarely passes first time
- Application changes frequently breaks the automation scripts
- There is a need to spend a lot of time and effort on automation maintenance

Question: Why does the automation maintenance take so long?

This is one of the biggest criticisms of automation: A minor application change can result in massive amounts of automation maintenance. Senior stakeholders just see lots of failing tests and their automation resources doing maintenance instead of delivering new value.

More often than not, this is reactive maintenance, where the application has changed and test code is subject to unplanned changed.

There are technical answers to this one around automation code and maintenance efficiency, but we're not doing technical in this article.

The softer alternative answer is to consider that quality is everyone's responsibility. Even in Agile cross-skilled delivery it's easy to silo people where the developers develop and the testers test. From an automation point of view, if there are unexpected changes that happen in a project (for example: changing of GUI IDs, or API data structure) don't immediately react and accept the changes in order to make the tests pass again. Instead, talk about it. Validate if these sorts of changes are expected within the team and see if it's worth the automation maintenance effort vs the actual change. A five minute development task can mitigate hours of automation maintenance.

If you have a lot of recurring maintenance pain, ensure your application is testable.

Story: At a client there were several Agile delivery teams and before I started, one team already had a shaky reputation. A recent story I had heard from the QA team was that one of the front end developers refactored an important page without a supporting story card. The unit tests and API tests in the CI pipes passed and it was pushed to the test environments. A couple of hours later a fair portion of the selenium GUI tests were no longer working. The QA team dropped what they were doing and spent hours fixing and rerunning the automation pack. It's commendable that the QA team reacted to verify the quality of the application and adjusted course accordingly, but this was substantial unexpected and (arguably) unneeded test effort.

Moral: Test maintenance will always need to be done – but in some situations best intentions from the QA resources to just keep tests running isn't always the best solution. It's worth a quick check with the team if the change is expected and still worth the maintenance effort.

It stinks that tests sometimes fail the first time and we rerun them over and over until they pass

The notable odours:

- The automation doesn't pass first time
- There is time spent rerunning tests
- Automated tests run in CI pipes then are rerun on local machines

Question: How often do the automation tests fail? - And do you know why?

I like this one as a definition of madness is doing the same thing and expecting a different result!

An automated test has run and failed. You glance at the error but decided to just press “run” again.

This is quite a common issue to the point where a number of testing frameworks have rerun-test-on-fail options.

I admit that I've done this at times and will continue to do so in the right situation. As a tester you have to complete the execution to get a deliverable over the line. You know the test works, running it again takes a few minutes and it'll probably pass. Perhaps the test machine hiccupped, there was a network issue or there was some other non-repeatable server issue. It's more efficient to press run and wait a few minutes then it is to spend an hour or two debugging.

While the creation of new work has the highest value there is a tipping point that means it is time to clear some technical debt in the test suite. Rerunning automation is potentially a short term plaster on bigger cracks. Good automation just runs.

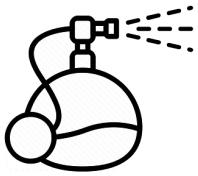
If you catch that you’re frequently re-running tests on fail, you need to fix the underlying problem. It's worth the investment to make sure you have a well synchronised automation pack and detailed

auditing capabilities. Having good reporting/tracking at the test level and consistent low level reporting lets you identify flaky tests and make dramatic improvements.

Story: I joined a client team who were executing GUI automation in a CI pipe with around a 40% pass rate and it had been at that level for a while. It was attracting adverse attention from managers who only looked at the high level stats.
A little bit of investigation determined that there were a number of technical issues that made the reliability of the cloud CI build machine very poor - but the application was *seemingly* fine. You couldn't deliver software with so many failing tests so the automation was run in the CI pipe then the QA resource(s) were re-running *everything* locally again for a regression run on every sprint. Essentially, the CI pipe results were running but being ignored - therefore the history of runs and audit trail was non-existent.
Over a few weeks we brought in some new software to standardise and stabilise the execution - but that wasn't enough. When the number of test failures went down to manageable numbers, we made a point of investigating and fixing **every** automated test fail.
The end result was 98% pass rate on the CI pipe - the final 2% we had defects logged. There were intermittent application issues around asynchronous calls that everyone assumed to be automation synchronisation issues - until we looked into them in detail. We could only look into them because there were a handful of failing tests per run.

Moral: Find time to invest in good automation maintenance and good synchronisation. Rerun on fail is OK to achieve deliverables but still investigate any intermittent issues. A healthy test pack runs first time and provides results you can trust.

It stinks that only one person knows how to run our automation



The notable odours:

- A specific resource needs to be available to run the automation
- Automation is configured to execute on someone's account
- Changing target test environments (or other configuration) requires a code-level change

Question: Who can run the automation?

Automation just needs to run and test execution should be the simplest part of the life-cycle.

As much as possible, automation execution shouldn't have intrinsic/complex knowledge on how to execute the test pack and therefore shouldn't rely on a specific resource.

Automation doesn't stop at the test script. It's good to abstract as much complication as possible out of the execution, have automatic reporting and allow anyone to initiate execution with non-technical steps.

Story: At a client site I was the only Automator, I was this resource limitation as no one else knew the tools involved. Knowing that I was on a time limited contract it was important for my professional reputation to ensure that once I left, people would be able to execute the automation and report the results. I created additional scripts to replace what I knew in terms of setting the target environment, resetting machines and reporting.

Moral: Automate everything in the execution process. Make execution of the automation as complex as one button (or no buttons if there's a CI trigger or a scheduled timer). If there is a test management tool (or equivalent) have it manage the key aspects of the execution (e.g. target browser, test environment, etc.)

It stinks that automation needs manual intervention

The notable odours:

- Automation has several packs that are kicked off with a manual air gap between them
- There are manual phases to the tests
- Manual interference is required to make the tests run

Question: Does the automation need any manual help? – is test data set up manually? Is there a manual clean down between test runs?

Watch out for manual steps in the automation flow.

Automation doesn't stop at the test script. It's generally good practice to automate all parts of the process, so once it's running no further input is needed. The automation utopia in execution is to aim for the tests to set up their own data, execute, clean up and report without the need for any interference.

Story 1: This story is not one I was present for but from a friend early in my career that stays with me to this day. Our consultancy was tasked with automating a call centre application. Part of the test required the customer to answer the phone so the application would capture that the phone call was successfully connected. The automation engineer programmed in his desk phone number - as the test ran, he would pick up and put down the phone every time it rang. It's a logical solution to an application but it's not automated.
The low cost solution was to allow the dialling application to make an incomplete call then update the database value to be a completed call. We were OK to make the direct database modification as the purpose of the test was not to make calls but rather for the integrated system to detect the customer had been contacted. The incomplete call created the relevant

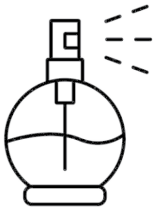
records and the automation just toggled the call-success flag. This removed the need for a tester and a desk phone.

Moral: Find the automated solution to application problems.

Story 2: I was part of a team scoping out some new automation work for a client. Throughout our engagement we found that the System Integrator had started automation but never finished. It was a commercial off the shelf system with a lot of integration points, however, the environment where they developed the automation didn't have the additional systems. Their automation would stop every time it reached any integration point and when the project reached the right environment, they would do those off-system steps manually. They had essentially baked in a large amount of execution complexity, framework knowledge and tool skillsets for a little time saving. The solution was for the client to ensure sufficient environment stubs in the short term followed by access to the full application landscape in the long term to deliver the highest end to end value.

Moral: It's not automated if you're doing manual steps.

It stinks that we have a lot more than one automation tool



The notable odours:

- There are multiple frameworks, multiple languages and/or multiple tools in play
- Job adverts list a few different test tools
- Specific resources are needed for specific projects

Question: What automation tools do we use?

Using a single tool aligned to the development language (where possible) is generally good automation practice. This has a number of benefits, such as minimising license costs, skill requirements, handover time, training effort as well as increasing the ability for cross project support for resources. That is what I stick to and when speaking to a client and it is still my recommendation.

However, I have experienced a notable caveat on this automation smell. There are some cases where multiple automation tools can be suitable. I worked with a substantial client who had hundreds of applications delivered by several third party vendors/teams across multiple concurrent projects over many years. Whilst the client's view was that "doing automation" was key, communication over such a landscape and resource availability with the same skillset was limited so they used different technologies to develop and different automation tools to facilitate their key requirements. Their automation return on investment was smaller, but it was still a positive value.

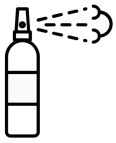
Story 1: During an automation health check, the third party system integrator had two versions of the same automation tool on site. The first (earlier) version was integrated and limited by the test management tool version. At some point they required a later version to support a new application under test, but this couldn't integrate with the test management tool. As such they built and operated two independent automation frameworks with a whole host of differences for execution and reporting.

Moral: Try and stick to one solution wherever possible. If the test management tool is the problem, change it. If it can't be changed, find another way. For example, create your own integration using a single framework. Try to avoid doubling the cost and complication.

Story 2: I joined an existing delivery team and the client direction was to make sure the teams invested in automation as they progressed through the project. The intention is that when you pick up the project again, you have a working regression pack before you make changes.
However, whether it was deliberate or not, they didn't go as far to say what tools had to be used. As a result, at least 4 different automation tools were in play - one of them was selenium and even this had C# and Java variants.
As they were buying tools from project budgets, getting a few thousand for test automation tool licenses or borrowing existing stock was not a major hurdle for the client.
On the one hand this is costly and inefficient as there are excessive licenses, there's a lot of tools to install and manage. On the other hand, they still had automation. The scripts were written with automation in mind, the automation was done well and the test data was correctly identified. The solution in place was working for the client but they were continually impacted with automation skillset constraints. It was only when a major modernisation program was kicked off that all these piece-meal automation tools were standardised to a consistent delivery approach.

Moral: Delivering good automation is important - more important than the tool selection. It's more costly to have a variety of tools but once automation is done correctly it can be migrated or resources can learn the required new skills. However, this must not be a free for all – there must be strategic control of the tools to prevent excessive overheads.

It stinks that we often see Automators watching the automation run



The notable odours:

- Key resource are sat at their desk watching the automation run
- Automation is running on scripting machines

Question: Why are the team watching the automation?

A couple of reasons for this one.

The first has a quick and simple resolution when you catch it. It's easy to fall into a BAU-status-quo:

- You start automation from scratch on your machine.
- There are only a few tests so when you press execute it takes a few minutes and you really need to babysit it at first to make sure it behaves as expected.
- Then you add more and more tests until the pack is takes hours to run.
- Eventually you lose your entire working day every time the pack needs to run.

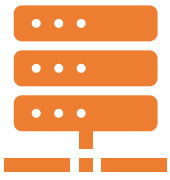
The second reason is that automation is running on scripting machines.

Typically, a scripting machine (the Automator's local desktop) has extra configuration, extra software and generally extra effort applied to it. All this can influence the application or the execution of the test.

When it comes to test execution you want it to run on "standard kit" where any configuration is handled by the automation code itself with no manual setup required. This allows automation to just run, creating a lower maintenance and more scalable solution.

Story: I was visiting an existing client site. On my rounds I stopped to speak to one of our consultants and at that point he was executing the regression pack. He hadn't created it but was the current custodian of it. It was running on his client workstation and as it was thick client GUI automation it blocked him from all client work. The pack had to be run in office hours as the machine needed to be unlocked. Pass and fail results were only delivered at the end of the run at the end of the day. In this instance, all it took was to flag to the consultant what he was doing and he recognised the impact. The next day he had a second machine which would remotely execute the tests and returned a day a week of his working time to the client.

Moral: Aim to execute formal tests runs on controlled machines, i.e. machines that are not used for scripting. This can be resolved by having more physical machines, virtual machines or executing the test pack from a CI pipe on build agents - anything as long as you can keep working while the automation is running.



A Final Note

Having automation that works is great. I would say that's a good start but we should always strive for continual improvement.

When you have automation it's good to think that:

- Automation needs to test the right things:
  - It needs to report the right information to the team and the output needs to contain trustworthy results.
  - Someone is paying for the automation. If they don't feel warm and fuzzy or hear bad things about the automation/project they might reassess.
- Automation needs to just run:
  - Consider that for every application change, automation needs to run. If it doesn't *just* run there is something there that is causing a waste of time and money.
  - If it needs manual intervention then, by definition, it's not automated.
  - It needs to run on remote machines with all configuration managed by the execution process.
  - You need the right tools for the job and avoid diversifying in automation tools where possible.
- Automation needs to run well:
  - If it's a lot of effort to run (for example it falls over or gives poor results) people hate it. If people hate it they'll find reasons not to run it or cut corners when doing it.
  - If someone can suggest improvements that can be made to the automation - make a case for them. Seek budget or time to make those key changes.
  - Automation maintenance should be proactive and agreed within the team where possible.

The best practice test automation solution varies from client to client. With the points in this article you don't need to be an automation expert, you don't need to be technical and you don't need to review any code to help the team achieve their version of best practice. By catching a hint of the odours or innocently asking some probing questions you can shine a light on any problems or impediments and the resolutions can become logical. That might be all that's needed to get them fixed by the right people and allow everyone to enjoy the sweet smell of success.

If you know you have complex automation smells and cannot discern the source of the odours then please contact info@shiftleft.today or call Alan on +44 (0)7469 702042 and we can help.