INFUSE

A GUIDE TO TEST AUTOMATION

A Guide to Test Automation

Who is this guide for?

This guide is written for anyone interested in implementing test automation, at any stage of project maturity using Agile, DevOps, Waterfall or Bimodal delivery methods. Its aim is to provide the reader with a handy guide on the subject that includes key considerations for the implementation of test automation.

Abstract

This guide explores various discussions and findings based on several roundtables hosted by Infuse at Next Generation Testing 2016, TestExpo 2016, and Agile Methods and DevOps 2017 conferences, combined with best practices from Infuse's test automation practice leaders and practitioners. The guide is structured around the Infuse Services Delivery Methodology (iSDM) and is written by several people at Infuse.

We also summarise the market trends on test automation. For example, a large divide remains between ambition and reality when it comes to test automation coverage: many strive for roughly 75% (non-unit) test automation coverage yet just 16% of testing activities are automated. To bridge the divide, we need better test automation people, processes, and tools (see the Test Automation Challenges and Test Automation Benefits diagrams sourced from the 2017-18 World Quality Report). Test automation may be difficult but it can generate superb results. Reducing test delivery times from a matter of weeks to hours or minutes is not uncommon when automating regression packs.

Why Automate?

Organisations decide to automate tests for many reasons, such as:

1) Cost - manual testing is resource-intensive and therefore, costly.
2) Time - manual testing cannot keep pace with certain tasks.
3) Accuracy – there can be a greater chance of error with manual testing many repetitive tasks.
4) Trend – many organisations have realised benefits from automation, so there is pressure to follow suit.
5) Scale – manual testing cannot match the complex iterations of automated testing.

Organisations are automating because there is an increasing need to support faster time-to-market and the demand for high-quality software releases is converging. We need, and demand, incrementally better software releases, and we need them delivered faster. One of the best ways to tackle this problem is with test automation. In fact, the number one recommendation in the 2017-18 World Quality Report is to increase the level of smart test automation. However, there are no shortage of key challenges in implementing automation (see Figure 1). The reality is that test automation is desirable but difficult.

A large divide remains between ambition and reality when it comes to test automation coverage: many strive for roughly 75% (non-unit) test automation coverage yet just 16% of testing activities are automated. To bridge the divide, we need better test automation people, processes, and tools (see Figure 1). Test automation may be difficult but it can be very beneficial (see Figure 2).

The decision to invest in automation must not be taken lightly as the risk of failure is real. There are many pitfalls to avoid. Some of the biggest failures we have seen are due to organisations trying to conform to what they think is the right thing to do. For example, we have seen some organisations try to automate everything. They have been misguided to strive for 100% automation, which is a foolish – and expensive - thing to do.

Test automation can take a significant amount of resources to implement, and yes, you will initially make mistakes. However, if you can the right mix of people, process, and technology, you can realise the many benefits of test automation.

Key challenges in implementing automation

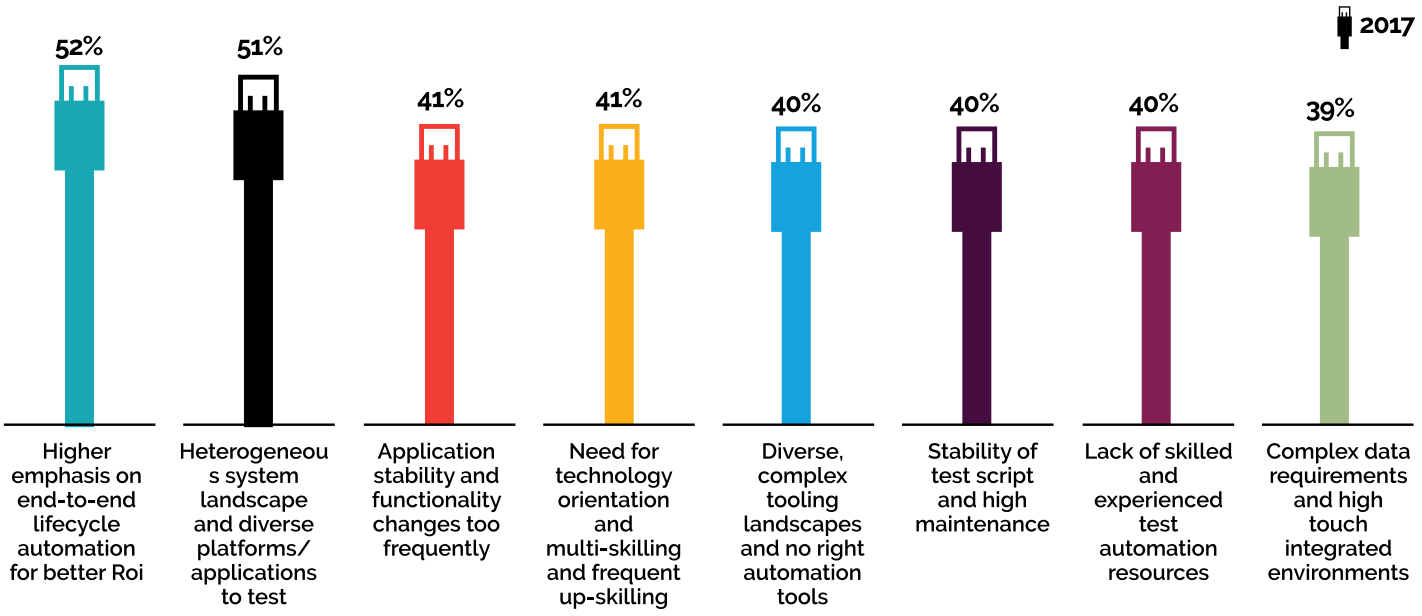


Figure 1: Test Automation Challenges (Source: World Quality Report 2017-18)

Benefits of test automation

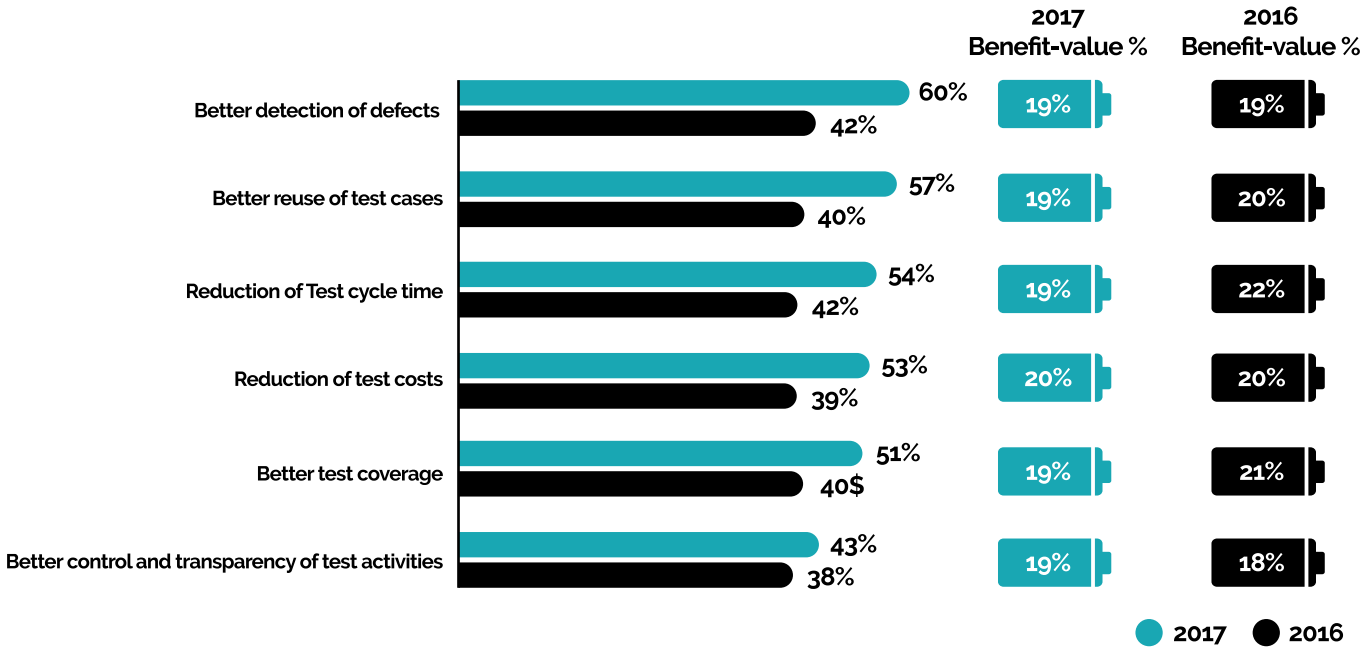


Figure 2: Test Automation Benefits (Source: World Quality Report 2017-18)

How do we get started? The iSDM Approach – Discover, Design, Build, Deploy, Realise

Automating will require discipline, process adoption, checklists, and entry/exit gates. It doesn't matter if it's Agile, Waterfall or DevOps; it needs a methodology.

At Infuse, we use iSDM, the Infuse Services Delivery Methodology, for delivering testing and iPMM, the Infuse Project Management Methodology, as our governance framework (each summarised in Figure 3 and Figure 4). You will need to consider tools when working out your process, but needn't – and shouldn't – select tools until you determine your process and needs.

iSDM - Discover

The Discover Stage is the first step in our approach to test automation delivery. If you are not working with third parties, there is still merit in this stage to clearly understand your motivations, goals, etc. with test automation. If you are working with third parties, during this stage, your partners should seek to obtain a thorough understanding of your business and define the project objectives through a statement of work. At this stage, we look to evaluate the client's needs by collecting information via questionnaires, conversations, interviews, documentation, workshops, and past experience.

Customer Engagement

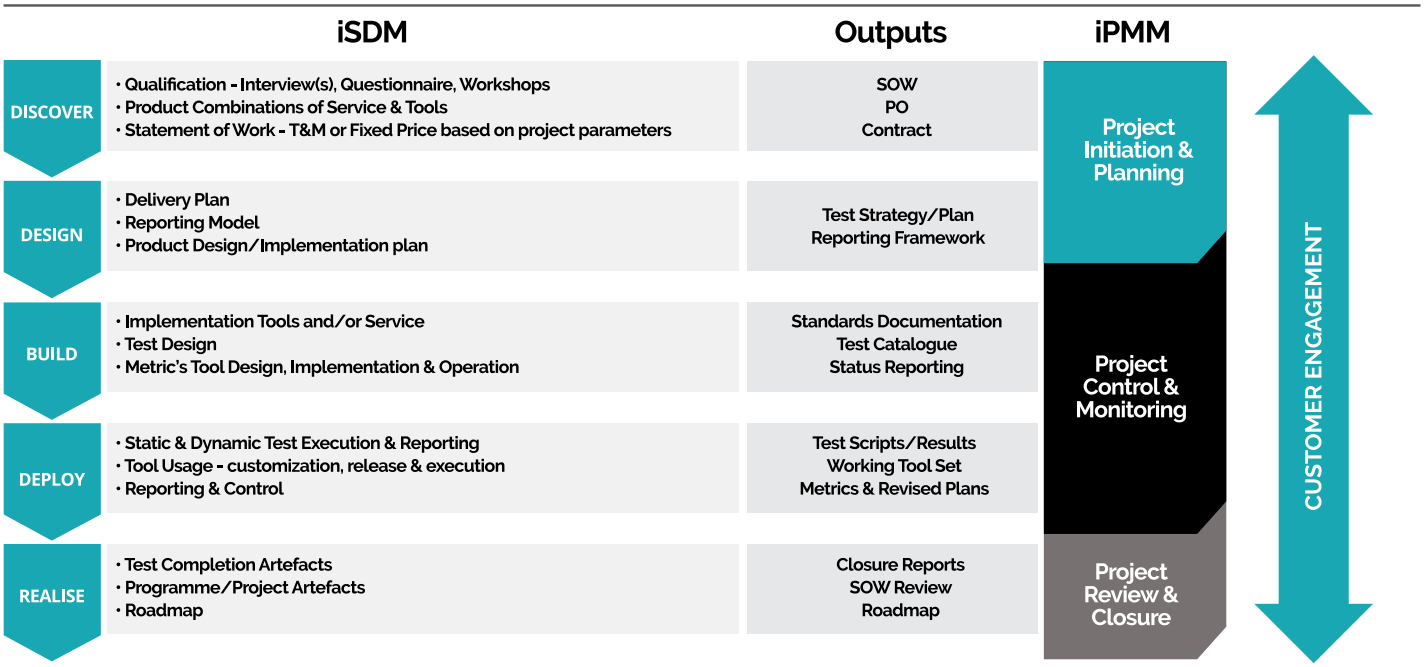


Figure 3: The iSDM Customer Engagement Model

High-level Infuse iSDM and iPMM process

DISCOVER	DESIGN	BUILD	DEPLOY	REALISE
Purpose To elicit and understand the client's requirements for the test implementation	**Purpose** To design test approach and test artefacts	**Purpose** To construct the test artefacts according to the design	**Purpose** To execute the test artefacts according to the plan	**Purpose** To demonstrate benefits of service and formally close assignment
Key Tasks: • Review PID • Setup RAID log • Determine baseline for ROI (and SLA's) during Relise phase •Conduct proof of concept •Sign off SoW	**Key Tasks:** • Review project deliverables • Automated Test Case Candidate Selection • Construct test suitability matrix • Build test strategy and plans and obtain sign off • Review test basis documents • Design test artefacts as per plan	**Key Tasks:** • Setup test environment including framework • Construct Test Cases • Generate and validate test data • Generate Components and Build Test Scripts • Review and approve Test Scripts • Gather and distribute Test Preparation Metrics	**Key Tasks:** • Baseline test scripts • Execute test scripts • Raise defects • Product and distribute test execution reports • Chair defect review meetings • Construct end of phase testing reports • Sign off test phase	**Key Tasks:** • Confirm objectives met, benefits delivered and value-add identified • Construct Project Completion Report and sign off • Archive test assets • Arrange Post Implementation Review (PIR) • Construct PIR Report
Mandatory Deliverables: • Statement of Work • Proof of Concept report • RAID log • Resource and milestone plan	**Mandatory Deliverables:** • Test Strategy • Test Plan (inc test case specification) • Test Conditions • Requirements Coverage	**Mandatory Deliverables:** • Test Environments • Test Cases • Scripts • Test Preparation Metrics	**Mandatory Deliverables:** • Test Execution Evidence • Logged Defects • Test Execution Metrics • Test Execution and Defect Reports • End of Phase Test Reports • Code coverage	**Mandatory Deliverables:** • Archived Test Assets • Project Completion Report • Post Implementation Review Report

Figure 4: Infuse Service Delivery Process (iSDM) with approach detail)

Build a business-case

If your investment in test automation cannot be supported by a solid business case, i.e. if there is not an identifiable return on investment (ROI), then you should not proceed. Key to building a business case for test automation is to answer the following key questions:

Current state analysis (AS-IS) OPEX:

- What is the number of manual test cases executed per run?
- What is the cost per person to run the test?
- How many people are running manual tests?
- How many times is it run per year?
- How many defects leak into production due to inadequate regression testing?
- What is the cost of resolving a defect in production?
- What is the cost of testing project slip per day?
- Calculate the Capital Expenditure (TO-BE):
- What is the tooling cost (licenses)?
- How much will it cost to automate the tests?
- Calculate the OPEX for automation:
- How much will maintenance cost per year to keep the tests working?
- How much will it cost to run it each time? (manual and automation)
- How many times will you run it in a year?
- How many tests do you need to automate?
- What is the cost of test tool license support and/or renewal per annum?
- How many defects can you prevent leaking into production in this approach?

Typical Assumptions we use in our business cases:

- Average project delay in testing – 5 days
- % reduction in regression defects – 80%
- % reduction in project slippage – 80%
- % test case maintenance – 10%
- % of tests' automated – 80%
- Cost for resolving a production defect - £2,500

Figures 5-7 show some anonymised output samples in a business case.

Manual Testing vs. Automated Testing Cost Comparison of 3 Years

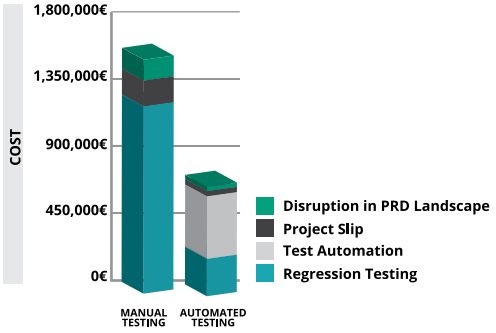


Figure 5: 3-year cost comparison of manual testing vs. automated testing.

Manual Testing vs. Automated Testing Cumulative Cost

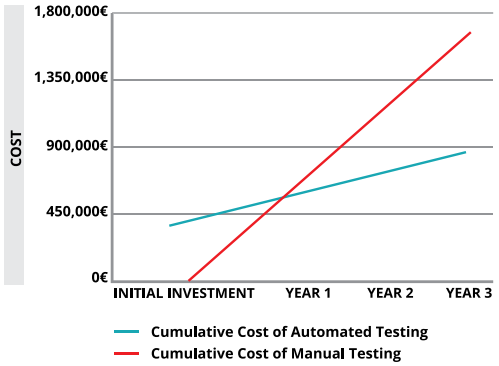


Figure 6: The cumulative costs of manual testing vs automated testing.

Cumulative Savings after Automated Testing

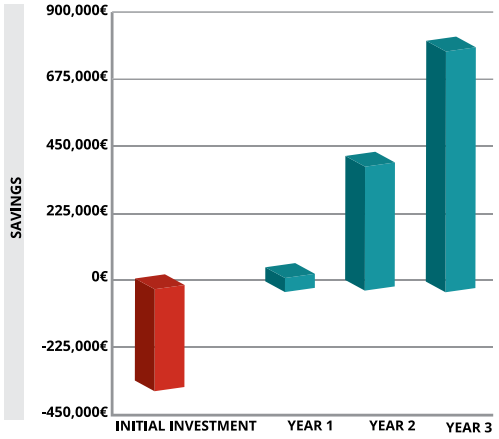


Figure 7: Cumulative savings of automated testing.

Find the right people to lead the way

Finding the right person that can strategically map out a course for automation is an important first step. Don't be too afraid to disrupt the status quo. Progress requires change, which means going against the status quo. Having at least one or two people on your team with a track record of delivering test automation can go a long way. Alternatively, partner with a third-party vendor that has the skills and experience needed to delivery test automation success.

Enterprise vs Open Source tools

An increasingly frequent problem is a rushed decision to ditch enterprise tools and adopt open-source tools without first working the business case. It may feel like it's cheaper to go Open Source but it's no guarantee. Baseline your business case during the Discover phase with a proof of concept (POC) or proof of value (POV) exercise to define your milestone and success criteria as well as confirm your estimations and assumptions.

Risk, actions, issues and decisions log

While building your business case and doing your POC or POV you will understand the critical risks behind the actions and decisions you have made. Keep this log well maintained when you need to justify your choices. While Agile places more value in working software than comprehensive documentation, it does not mean no documentation.

Focus on the foundations

Start your automation efforts early in the process, at the unit testing level. Get developers to write automated unit tests. It's the foundation of good test automation and key to the test automation pyramid (covered later).

Use the right people the right way

It's a common pitfall to think that anyone can automate. Don't underestimate the technical aspects of test automation and the skills that they demand. Ensure your tools and framework support the right people in your organisation. Not everyone will become an automated test engineer but that doesn't mean your non-technical testers cannot be involved in the automation. A good framework will allow you to leverage your non-technical testers to deliver test automation.

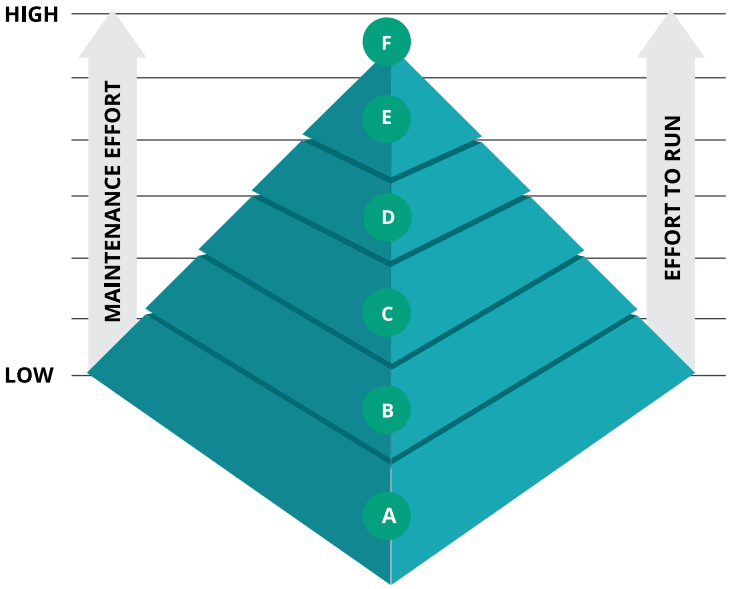
Vary your automation approach per the task at hand

You may need a different test tool for different test levels in the testing pyramid.
For example, with microservices, you don't need to worry about UAT testing.
However, you do need to think about API testing and service and network virtualisation.
Think about the right technology or framework to handle your automated testing pyramid.

Follow the concept of the test pyramid

Aim to have many more low-level unit tests than high-level end-to-end tests running through a GUI (strive for at least 80%-unit test coverage). While high-level, end-to-end tests can be brittle, expensive to write, and time-consuming to run, you still need them as a second line of defence. We believe having the right tool and framework (such as our useMango functional automated test tool and framework) can combat these issues.

The Test Pyramid



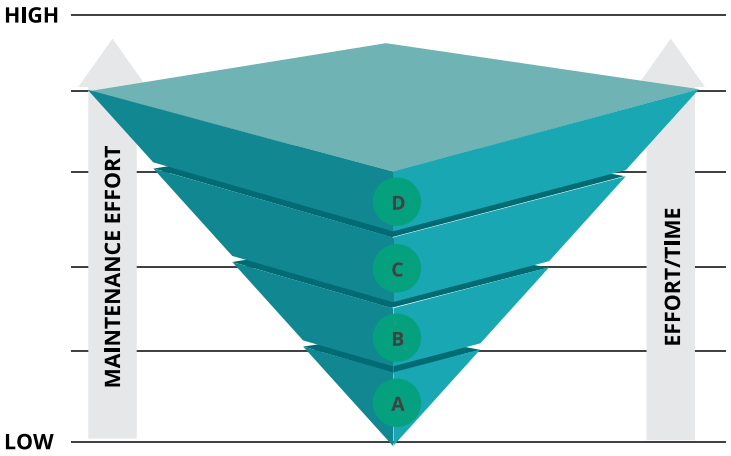
A. Unit
B. Component
C. Integration
D. API
E. GUI
F. Exploratory

Figure 8: The Test Pyramid

Determine what part of the software test pyramid makes sense to prioritise

Some start with unit tests while others start with GUI tests. There's no one right place to start. Just start reducing the test triangle of debt as soon as possible. Consider your own organisational need and start there.

The Test Pyramid of Debt



A. Unit
B. Integration
C. GUI
D. Manual

Figure 9: The Test Pyramid of Debt

Mandate Test-Driven Development (TDD)

Developers are under time pressure to deliver code so if testing isn't part of their requirements, it may not happen. One way to increase unit test coverage is to mandate developers to use TDD. This can come from Project Managers or Programme Managers if they have the authority to do. If developers aren't expected to write unit tests, defects will inevitably occur and may not be found until much later in the development lifecycle, which will become very costly and time-consuming to fix

Just do it

Starting is half the battle. Testers can be nervous at first when it comes to automation. It can feel like you're losing control of your work to developers (of writing a lot of test scripts or learning a new tool).

iSDM - Design

The Design Stage is the process of creating an optimised solution from the knowledge and information that is gathered at the Discover Stage.

Review project deliverables

Review your project deliverables to ensure they can be delivered and tested (e.g. stories, design documents, sprint plans, roadmaps and acceptance criteria are achievable).

Requirements coverage

Demonstrate the percentage of requirements covered by tests, including their latest execution status, using the test management tools you have.

Test coverage

Demonstrate the components impacted, using the test management tools you have.

Test strategy

Your test strategy should cover the approach you will take. It should be pragmatic and follow a certain delivery methodology. Remember that software build processes are rarely completely repeatable, and many mistakes are made during the integration and build stages. Do not underestimate the need for intelligent regression strategies.

Test plan

Plans are needed to show the scope of testing at the systems Integration phase, to include components/systems impacted, environments to be used, tests planned, resources required and reporting mechanisms.

Which regression tests should I automate?

When striving to increase coverage, perhaps to 85% (non-unit) test automation coverage as many desire, you need to decide what to automate. Here are some key considerations:

Start small
Start with mundane, daily tasks that are repeatable. You can learn this through results analysis.

Only cover as much as you can maintain
This rule of thumb helps you focus on the basics. For example, automating logins or registrations.

Choose shorter, smaller tests
In general, the bigger the test, the more unreliable the test is. Small tests are better to automate because they are more reliable.

Choose tests with relatively low amounts of interdependence and decouple where possible
If something goes wrong at the start of an automated test with interdependence, so, too will the rest of the test. Create tests that are independent of each other.

Keep it simple
Automated tests are code, and like all code, the more complicated it becomes, the greater the risk that it will contain bugs. Think of regression tests that you know are simple and repeatable. Add complexity to test automation only after you've established a baseline of simple automated tests.

Select tests that matter
Ask yourself, if the test fails, does it matter? If yes, how much? This is key to the prioritisation of tests to automate. For example, a login failure is a critical failure whereas a slightly altered logo file is not. Also consider the business case, i.e. the payback on what you're automating. Does the benefit outweigh the investment cost (time and money)? Generally, the rule we use is if you run it more than two or three times, automate it.

Ensure that you are building the data underneath the test automation to support it
Tests are only as good as the data that drives them within the execution. Think about the data to create, read, update and delete so the process is repeatable. Think about the data that will also drive the actions on the test.

How do I prioritise tests to be automated?
You now have a list of tests to automate. Likely, a lengthy list. Consequently, you now need to prioritise the automation of these tests to get some quick wins and prove the value of test automation.

In addition to the points outlined below, Francis Miles of Infuse has provided his thoughts on the subject in a LinkedIn post here.

Always use a risk-based testing approach
There are a few things you need to consider in terms of choosing parameters. Let's start with the probability of failure and frequency of use as outlined in Figure 10:

How to prioritise tests: Probability of failure vs. Frequency of use

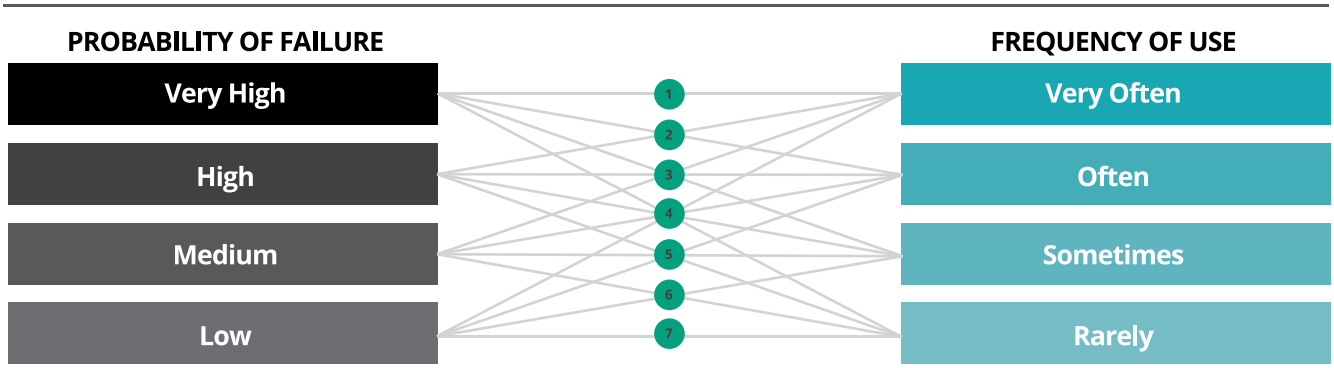


Figure 10: How to prioritise tests: Probability of failure vs. Frequency of use

You then feed these categories into a test case selection model as outlined in Figure 11. This can then help you prioritise your tasks.

Risk Analysis Example

Test Name	Type of Process	Weighting	Impact of Failure	Weighting	Affected Users	Weighting	Number of Transactions	Weighting	Weighed Average
Test 1	Display	1	Legal	3	Many	3	More than 3	3	2.5
Test 2	Calculation/Validation	3	Wrong information	2	Some	2	Between 2 and 3	2	2.25
Test 3	Update	2	No impact	1	Few	1	2 or less	1	1.25
Test 4	Display	1	No impact	1	Few	1	More than 3	3	1.5
Test 5	Display	1	No impact	1	Few	1	More than 3	3	1.5
Test 6	Calculation/Validation	3	Wrong information	2	Many	3	More than 3	3	2.75
Test 7	Update	1	Wrong information	2	Many	3	Between 2 and 3	2	2
Test 8	Update	1	Wrong information	2	Some	2	Between 2 and 3	2	1.75
Test 9	Update	1	Wrong information	2	Some	2	2 or less	1	1.5
Test 10	Calculation/Validation	3	Legal	3	Many	3	2 or less	1	2.5

Risk Analysis Example (Prioritised)

Test Name	Type of Process	Weighting	Impact of Failure	Weighting	Affected Users	Weighting	Number of Transactions	Weighting	Weighed Average
Test 6	Calculation/Validation	3	Wrong information	2	Many	3	More than 3	3	2.75
Test 1	Display	1	Legal	3	Many	3	More than 3	3	2.5
Test 10	Calculation/Validation	3	Legal	3	Many	3	2 or less	1	2.5
Test 2	Calculation/Validation	3	Wrong information	2	Some	2	Between 2 and 3	2	2.25
Test 7	Update	1	Wrong information	2	Many	3	Between 2 and 3	2	2
Test 8	Update	1	Wrong information	2	Some	2	Between 2 and 3	2	1.75
Test 4	Display	1	No impact	1	Few	1	More than 3	3	1.5
Test 5	Display	1	No impact	1	Few	1	More than 3	3	1.5
Test 9	Update	1	Wrong information	2	Some	2	2 or less	1	1.5
Test 3	Update	2	No impact	1	Few	1	2 or less	1	1.25

Figure 11: Test case selection model

Size
As a rule of thumb, the smaller the test, the better.

Business importance
Are you automating the tests that are of interest to your end user and therefore, your business?

Data
Prioritise tests that have reliable data. More on this later in the document on test data strategy.

System Under Test
Testing from a command line of API is easier. Legacy systems, in general, are harder to automate tests due to outdated UIs. The longer the life cycle of the system under test, the more valuable automation becomes because it is re-run multiple times.

Maintainability
How maintainable are the tests? Are you trying to automate something that will change?
What is the cost of change?

Avoid the pitfall of trying to automate everything
The earlier you automate testing in the process, the less need for automation later in the process. Start with unit testing in development, if possible. In Agile, test-driven development demands developers to use unit testing as they develop to address quality early in the process. These unit tests are written before the code, which is far earlier in the process than when there are many tests. This requires testing throughout the development process, which helps with defect leakage and enables defect prevention.

iSDM Build

At the Build Stage, the build deliverables are implemented in accordance with the design specification. Before building out your test automation, you ought to first invest in a test automation framework to keep pace with development and customer demand.

As a reminder, a test automation framework is an integrated system that sets the rules of automation for a specific product. This system integrates the function libraries, test data sources, object details and various reusable modules. These components act as building blocks that need to be assembled to represent a business process. The framework provides the basis of test automation and simplifies the automation effort.

The testing framework is responsible for:

- Defining the format in which to express expectations;
- Creating a mechanism to hook into or drive the application under test;
- Executing the tests;
- Reporting the results

Setup test environment including framework

What type of framework do I build? Do I build a page object model, hybrid, action-based, or keyword based framework? The problem with these choices is that they are all combinations of 2nd, 3rd and 4th generation approaches and therefore, still require a degree of maintenance.

The keyword-based approach ensures that the maintenance is performed at the keyword level. Any change to a keyword is reflected in all the places where it is being used. This is still quite a low-level approach to building reusable assets, though, and can be further abstracted to page object models by combining keywords across a web page or form and still require maintenance. The frameworks approach, however, simplifies the process of test case creation, allowing non-technical subject matter experts to focus on creating high-level business process test flows, while test engineers concentrate on enabling test automation.

Infuse also has a test framework and tool useMango™, which enables your testers to generate more page objects automatically with a scan in seconds. Where useMango cannot scan, a test engineer can create this in their chosen test tool (e.g. Selenium WebDriver or UFT) and make this available via useMango™.

How does test data factor in to test automation?

You now know which tests you will automate and in which order you will automate them.
Now before you start automating tests, you need to consider the data that you will use to drive them.

Here are some key considerations for test automation data:

1. Each test should be independent of other tests and baselined datasets.
2. Ensure baseline datasets are source controlled.
3. Develop a custom API to perform CRUD (Create, Retrieve, Update and Delete). Do not rely on test ordering to perform CRUD with multiple tests.
4. Use open-source libraries, such as Faker, to generate your own realistic looking data or purchase a tool
5. Don't only update and check UI. Always perform layer checks (i.e. Database) to ensure that a record has been updated correctly.
6. Use an API to setup, configure and teardown.
7. If you have the budget, invest in a test data management tool

In addition to the above considerations, there are all the test data strategies to consider, which are listed in Figure 12.

Test Data Strategies and their Impacts

Proposal	Test Development TIme	Difficulty of Test Creation	Test Execution Time	Levels of Realism of Data	Rework of Tests	Impact on other Test Teams
Create Data within Test	HIGH	HARD	LONG	LOW	LOW	LOW
Data Resets	LOW	SIMPLE	SHORT	HIGH	HIGH	HIGH
Batch Data Creation	MEDIUM	MEDIUM	MEDIUM	LOW/MED	LOW	MEDIUM
Find/Make Data within Test Data Management Tool	LOW	SIMPLE	SHORT	HIGH	LOW	LOW

Figure 12: Test data strategies and their impacts

Figure 12 also shows the impact of each strategy on the efficiency of test automation:

- Creating data and data resets are diametrically opposed in the advantages and disadvantages
- Batch data creation provides a compromise between strategy 1 and 2
- Finding and making data requires investment in tools and utilities. It is generally our recommended approach but it needs to be supported by a business case.

Other technical considerations for test automation

Now you're almost ready to start automating.
But before jumping in, it's helpful to have these best practices in mind:

1. Apply DRY and DAMP principle (DRY - Don't Repeat Yourself and DAMP - use Descriptive and Meaningful Phrases) to improve reusability and Readability.
2. In general, high volume, high risk tests are the best candidates for automation.
3. Reduce overlap and duplication from the first day. Once you let duplication creep in, it will get worse and become more expensive to remove.
4. Thoroughly understand the data flows.
5. Always work on the principle that automation is to cover risks and report on the state of the system quickly.
6. Do not try to automate everything. For example, checking PDF outputs can be done outside of core automation.
7. Treat test code as production code.
8. Be thoughtful.
9. Ensure that tests are readable. Readability is even more important than the correctness. Unless you can read a test clearly, you will not be able to judge whether it is testing the right thing.
10. Aim to run all your automated tests quickly and reliably, ideally as part of your build/CI process.
11. Automation people are not just automation developers. They are also testers and they should be testing, not spending all their time on fixing broken tests.
12. Abstract shared and common functionality into classes. This allows test automation code to be written quicker, look cleaner and be easier to maintain.
13. Split function libraries based on functional areas (Excel Functions/Data Table Functions/ SQL Functions etc). Monolithic function libraries without logical structure or organised content can be a nightmare.
14. Comment functions within libraries thoroughly. At Infuse, we usually create HTML documentation for all our function libraries. This provides user friendly documentation for all functions and classes with very little effort (1 batch file call with 3 arguments documents everything).
15. If there is something not natively available in your tool, create it. Whether it be a class to extend the logging and checkpoint functionality (before-during-after checkpoints, custom logging, element/object specific checkpoints) or function libraries to cover functionality not available in the native coding language – there is always an option.
16. If you are not sure of a business process, it is vital that you ask. Time spent automating the wrong logical flow is time and money wasted. In almost all cases business users will jump behind automation, as it saves time and money, and will happily assist in any way they can to get it up and running.
17. If any code is duplicated in a test script, it shouldn't be there. Duplicated code should be abstracted to function libraries or classes.
18. Comments! Comments! Comments! One cannot stress enough how important well commented code is. All scripts should be commented throughout, as well as commenting of code in function libraries.

What are some of the more common problems with test automation that ought to be avoided? In addition to best practices, it also helps to be mindful of these common pitfalls with test automation:

1. Too much time spent on fixing broken tests.
2. Not keeping automation assets clean.
3. Not having started out with the right framework.
4. Slow and flaky tests.
5. High maintenance framework and tests.
6. Silos can kill automation efforts.
7. Avoid "whack-a-mole" testing by getting unit testing right. (Whack-a-mole testing occurs when you fix one issue only to find another pops up because you can't get to the root of the problem).
8. Picking the wrong tool.
9. Inadequate planning and expectation management.
10. Relying on the wrong people.

iSDM - Deploy

The Deploy Stage is where the assets created in the Build Stage are deployed.
Once you have built your tests you are ready to execute and start testing but before you do that:

Baseline test scripts

Execute your tests against the application three times so you have a baseline set of results to compare against.

Execute test scripts

Now you can execute your tests and see the productivity gains.

Gather and distribute test preparation metrics and test reports

Don't forget all that testing needs reports. Decide what you need to report.
Typically, at Infuse we'd report at least the following items:

- **Progress:** Reporting against the baseline to show if testing is making progress and identify areas of risk that may prevent the testing from meeting its schedule.
- **Quality:** A set of reports that show overall quality of the build and the coverage of testing against requirements. It allows for a risk-based view on the deliverable to be taken from factual information.
- **Transformation:** A set of reports that show the value that testing automation has brought to the project/programme and reports that identify the risk areas and investment areas in respect of environment availability and quality.
  These reports are detailed in Figure 13.

Key Test Automation Report

Report	Category	Calculation	Benefit
Test Execution Progress	**Progress**	(Cumulative number of test conditions actually executed / Cumulative number of test conditions planned to be executed) * 100	Shows the cumulative position on the progress of test execution. It compares the actual execution of test conditions against the plan Key tool for assessing whether the testing activity is on schedule
Defect Fix Progress	**Progress**	(Cumulative number of defects fixed up until the end of the reporting period / Cumulative number of defects detected up till the end of the reporting period) * 100	Demonstrates how many of the detected has been fixed and how many have remained open Shows elevated risk areas and where and when the defect fix progress falls behind defect detection
Age of Defects	**Progress**	Number of defects by time closed in a pre-defined time period (t < 1 day; 1 days < t < 3 days; 3 days < t < 7days; t > 7days) Calculate separately for severity 1 and 2 defects	How quickly are defects fixed, allowing for estimating future projects
Test Script Pass Rate	**Quality**	(Number of test scripts that pass at the first execution attempt in the reporting period / Total number of test scripts attempted to be executed in the reporting period) *100	Key indicator of build quality Useful to validate planning/estimating assumptions Powerful "early warning" mechanism for testing success or failure
Defect Detection Percentage	**Quality**	(Total number of defects detected in the lifecycle stage / Cumulative number of defects detected up till the end of the lifecycle stage) * 100	Shows the elevated risk areas Demonstrates how effective testing has been for each phase, particularly the effectiveness of third party supplier testing Indicates effectiveness of stage containment 80% of all defects should be found up to the end of the previous test phase
Traceability of Requirements	**Quality**	(Number of requirements that have been mapped to test conditions / Total number requirements in the project) * 100	Demonstrates that the project has delivered all that it has been asked to
Testing Defect Root Cause Analysis	**Quality**	Number of defects due to a predefined Root Cause (Requirements; Design; Build; Test preparation; Test Data; Test Environment; Other Root Causes) Calculate separately for severity 1 and 2 defects	Provides information that is a key input to deciding priority areas for process improvement and investment
Level of Automation	**Transformation**	(Total number of test conditions that have been automated during the project / Total number of test conditions for the project) * 100	Provides information input that is key to demonstrating the value of testing automation
Availability of Environments	**Transformation**	[(Total planned testing man hours for reporting period - Testing man hours lost due to environments not being available in reporting period) / Total planned testing man hours for reporting period] * 100	Supports the identification of the elevated risk areas and investment decisions around environment availability
Availability of Environments	**Transformation**	[(Total number of severity 1 and 2 defects - Number of severity 1 and 2 defects due to Test Environments) / Total number of severity 1 and 2 defects] * 100	Supports the identification of the elevated risk areas and investment decisions around environment quality

Figure 13: Key Test Automation Report

Chair defect review meetings

A defect review meeting is also commonly called triage, from the French word meaning to sort. At a minimum, the triage process should validate defect severities, make changes as needed, prioritise resolution and assign resources.

How do we ramp up?

After you have automated the basics, it's time to start thinking about ramping up your automation efforts with these considerations:

Don't try to automate everything – find the right amount of automation.

Automate what makes sense. As it is unprincipled to attempt to test everything, so, too, is it to try to automate every test. As a rule of thumb, if you run the test 2 or more times, automate it.

Get a test automation tool

Yes, we are biased because we offer a test automation tool and framework (useMango). But scripting all automated tests is very difficult to scale because scripting is a highly technical activity, requiring high maintenance levels, and presents challenges sourcing the right people because qualified testers that can script well are scarce. There are pros and cons of open source and commercial software.
In general, open source tools provide greater flexibility at a lower cost and commercial tools provide better functionality and scalability. When making the business case for a test automation tool, determine whether your organisation prefers the more traditional Capex approach (lump sum payment upfront for software) or the increasingly popular Opex model (monthly or annually recurring payments).

Build your test automation framework

Different frameworks work for different test levels. For example, you would have one framework for UI and another for API. Think of your test automation framework around your test automation tools. Tools like Selenium, JIRA and UFT are built for varying degrees of technical proficiency.
You can leverage a tool like Selenium if you know how to code. Development can help you with an automation framework and build out your test automation framework as you build out your tests. Infuse leverages useMango™ as the framework across different tools where applicable.

Shift team structures

Shift from siloed development and test teams to smaller product teams with both developers and testers. This Agile approach can enable automation because developers and testers need to work more closely and for testers to maintain pace with development, automation helps.

Adapt testing to delivery teams

With every new feature released, the time for testing increases if you can't automate the test. To increase coverage, one option is to include end-to-end testing as part of your definition of done (in an Agile model). Avoid the pitfall of just automating the latest feature without finding the time to automate the basics.

Find other places for automation

Automation can be applied beyond testing the product or system. For example, you can automate testing the process. In an Agile team, you may need to get involved in other things. Automating delivery through continuous integration and continuous delivery and using service and network virtualisation.

You can also use containerization to spin up environments to enable test automation without the need for real environments, which can be expensive and time-consuming to setup.

Use Behaviour-Driven Development (BDD)

We addressed Test-Driven Development earlier (TDD). Behaviour-driven development is an evolutionary step from TDD that combines TDD with the interests of the business.

Embrace DevOps Pragmatically

The full benefits of test automation can only be realised if you deploy your app frequently. Agile development with DevOps can enable frequent releases. Changing your process to Agile and DevOps and then adding Agile tools like JIRA and DevOps tools like Docker will help.

iSDM - Realise

The Realise Stage is where the benefits of the delivery are demonstrated. At Infuse, we use the Realise stage to validate the engagement has ended and to confirm we have met the contractual goals. From a test automation delivery perspective, a few things should be enveloped in the Realise Stage:

Confirm Benefits are met

As part of the project completion report, confirm objectives of the project have been met. This will involve identifying & communicating the business benefits of the project. Refer to the benefits baselining task that was carried out during the Discover stage. (If benefits will not be realised until future date then ensure a process is in place to capture this).

Post Implementation Review (PIR)

After every test automation project, a demonstration should be made to all key IT and non-IT stakeholders with a demonstration of the original business case, the technology developed and the lessons learned.

Typically, the areas that should be covered in a PIR are:

Area 1: Project Performance - Identifying whether the project:

- Delivered the business benefits
- Achieved the objectives specified
- Remained within the scope defined
- Produced the deliverables defined
- Completed within the planned project schedule
- Delivered within the budget defined
- Met the forecast resource levels defined

Area 2: Project Conformance: The extent to which the project has conformed to the relevant methodology adopted

Area 3: Project Achievements: Identify the major achievements of the project and describe the positive effect that each achievement has had on the business (including ROI)

Area 4: Project Issues: List any project issues and describe the effects they have had on the business

Area 5: Lessons Learned/ Recommendations: Describe the lessons learned from undertaking this project and list any recommendations for similar projects in the future.

Is the prevalence of test automation driving the increased use of the title Test Engineer?

There was a tangent at one of the roundtables that we hosted on test automation about the shift of many testers' job title from test analyst to test engineer. Some view the change as meaningless; just another title to describe the same job responsibilities. Others view it as a meaningful change for the role of the tester.
They argue that Analysts provide a detailed examination of the elements or structure of something whereas engineers design, build, and/or maintain structures (or engines or machines). As we discussed during this roundtable, 3 key questions for test automation dovetail with the 3 key responsibilities of an engineer:

1. How do we start automating = designing?
2. How do we ramp automation = building?
3. How do we sustain and maintain automation = maintaining?

Therefore, we would argue that the more testers become involved in test automation, the more apt the title of test engineer becomes.

Conclusion

We hope you enjoyed this guide to test automation. It covered a lot of ground: how to get started with test automation, determining which tests to automate, prioritising the tests to be automated, and effective use of test data. It also covered several key considerations for test automation: test case statuses and reporting, common pitfalls to avoid, and how to ramp up test automation.

In our view, there are several points worth reiterating. Firstly, organise your process. While it's tempting to jump into automation tooling, there is no point in automating a flawed process. Secondly, build a solid business case for test automation. It's an investment to make test automation work, so make sure your organisation understands the value of it. Thirdly, find good people/partners with experience. Automation is complicated, so be wary of people that only know enough to be dangerous. Fourthly, invest in an automation tool and framework to facilitate your automation efforts. Lastly, we at Infuse have done a lot of work in test automation and can help you with all facets of it: planning, training, tooling, and delivery.

To learn more, reach out to us today at info@infuse.it

useMango™ is a functional automated test tool and test automation framework that offers the benefits of test automation at a lower risk compared to traditional test automation approaches such as accelerator, keyword or action word frameworks. Designed with pre-built agile libraries/components, multi-platform Inspector tools, tooling for automation assets management and integration to modern ALM tools, useMango™ reduces both test execution time and cost.

Learn more about useMango™

ABOUT INFUSE

Infuse is a UK software testing company that provides modern software testing, transformation consulting and test environment management. We specialise in test automation and performance engineering.

Our strong alliance and partner network enables us to deliver the right solution for every client. Infuse is a Micro Focus partner in Application Delivery Management (ADM), an Oracle Gold Partner in Application Quality Management (AQM), a CA Partner in Dev and Test and SAP Partner. We have a global partner network to enhance our delivery capabilities beyond that of a typical UK software testing company.

To learn more about Infuse, please visit **Infuse.it**

INFUSE

For more information, email _info@infuse.it or visit infuse.it

Infuse Consulting Ltd | QEII Conference Centre
Broad Sanctuary | London SW1P 3EE | UK
Tel: +44 (0)20 3755 5135

facebook.com/infuse-consulting
@infusegroup
linkedin.com/company/infuseconsulting

infuse.it