



PLANIT GUIDE

Test Automation Fundamentals

Platform Tm Framework

Why Automate?

Tool Evaluation and Selection

Return on Investment

Automation Technical Concepts

Automation Tools

Automation Frameworks

Data Generation

Introduction

The 2015 Planit Index had this to say about Test Automation: “the number of organisations applying test automation has increased significantly, climbing 15% [over three years]. In some of the most active development industries including financial services, telecommunications and the software development industry itself, over 40% of organisations are applying automation as a key part of their software development lifecycle”.

Why is this? You only need to look at some of the key changes and trends to understand why. The rise of agile methodologies is one. Iterative practices mean a greater requirement for Automation in order to speed up regression testing. The ability to understand, execute and make amendments to existing automation suites (though not necessarily to create complex automation frameworks) is important for an efficient Agile team.

As Software testers, we need to be increasing our skills and our knowledge, so that we are capable of adapting to the new trends. We need wider knowledge, and deeper skills. Automation is one of these areas we need to focus on.

This guide aims to make clear what the benefits and pitfalls are when using automation.

This guide provides information on the following:

- What is Test Automation?
- When and why automation is recommended
- The Platform 4Ds framework which is a structured approach to the whole test automation lifecycle
- Automation Best Practice
- Behavioural Driven Development (BDD)
- Automation tool evaluation and selection process
- Where and how to automate, whether this is at User Interface level or at a business logic level
- What automation frameworks are and why they are important
- What tools do and descriptions of market leading tools
- How to calculate Return on Investment for automation and why this is important
- Test Data Generation using automation

Automation is not a silver bullet

Test automation can provide great benefits to an organisation and it is tempting to see it as a silver bullet for reduced cost, increased speed, and increased overall quality. However, it is important to be realistic about the challenges and commitment that is needed to deliver these goals.

Automaton requires considerable investment up front. It should be regarded as a development project in its own right which will need justification through a business case. It requires a plan, analysis, design, implementation and testing.

Automation requires continued maintenance. Automation tests must be regularly executed and maintained if the application under test is updated regularly! Each change applied to a system requires a consequent change to ensure that the automation pack still works. There needs to be continued commitment to having further development of the Automation solution.

There will always be a need for good manual testing. Some things just aren't worth automating; applications which change constantly or have a limited shelf life are not commercially viable. In addition, manual testers can perform exploratory tests that drive out hard to find or unusual defects in a way that automated test packs couldn't.

What is Test Automation?

“The use of specialised software that automatically executes the test steps to verify and report the test objectives against the requirements.”

Automation Scope

The focus of Test Automation has changed. It is no longer the case that automation is solely applied by replaying generic user interactions with the screen. Automation could now be used to test web service requests and responses through messaging gateways such as SOAP and REST. Tools may also be used to check the usability or consistency of the user experience across different platforms and to test databases, tablets, mobile phones and wearables.

Test Automation typically focuses on the areas below:

API Testing – This is testing focusing on Interfaces and Integration, to ensure accurate and complete transmission of data. This typically falls into System and Integration Testing.

Graphical User Interface – Testing of the observable behaviour in an application for user interface events such as keystrokes. This type of testing can fall across different testing layers from System, Integration and Acceptance Testing.

Data Source/Destination Validation – This includes testing the correct correlation between inputs and outputs. This can include databases, spreadsheets, csv and XML files.

Automation Testing and Tools

Test Automation Requires Test Tools. There are literally hundreds of Test Automation tools on the market, which range in compatibility, complexity and price. A review of the tools sector by type is included to illustrate the variety of options in the market. Specific examples are shown to highlight the capabilities of tools. These include:

- **HP Unified Functional Tester** – Long Time Market leader, formally known as Quick Test Professional
- **Selenium** – An open-source and very popular tool, with code-based scripting capabilities
- **Tricentis Tosca** – A Business-focussed Automation tool with internal logic that does not use scripting
- **SmartBear SoapUI** – A tool to automate API Service Testing, where the technology uses REST and SOAP protocols

Automation Testing and Data

There are many potential ways to automate test data, from manipulation of existing production data through to ad-hoc generation. Varying methods can be used to carry

this out from custom programming to specialist tools.

This guide provides options for data generation including:

- Application or GUI
- Sub Screen using HTTP
- Service Automation
- Database automation

Automation Testing and Risk

All testing should be risk-based. Functional testing covers the risks that will result in the system not functioning as required, based on business priorities and the likelihood of failure. Risks matter to a business. Risk is the possibility of a negative or undesirable outcome or event. Testing is concerned with two main types of risks:

Product or quality risks

Product risks are problems that can potentially affect the quality of the product itself, such as a defect that could cause a system to crash during normal operation.

Project or planning risks

Project Risks are problems that can potentially affect overall project success, such as a staffing shortage that could delay completion of a deliverable.

There are a number of types of risk that software can create:

- **Organisational**, such as efficiency loss and staff turnover
- **Financial**, both direct and indirect financial loss
- **Reputation**, loss of existing or potential customers and negative publicity
- **Legal**, in particular the failure to comply with legal or industry standards

So where does automation fit in? Risks are arguably greater now than they were.

- Systems are more complex (the weakest link in a chain of systems can bring the platform down)
- Technology is faster-moving and implementation lifecycles are shortening
- Users have higher standards and less patience
- There is a greater variety of platforms that can be accessed for the same functionality. Mobile Devices and operating systems versions are proliferating
- There is more outsourcing of solutions; virtualisation of systems and platforms and system boundaries are becoming blurred

These risks lead to increased requirements towards improving the quality, speed and accuracy of testing.

Define Making the Decision to Automate

Key Reasons to Automate

There may be a number of reasons why automation may be considered:

- The cost of manual execution is too high and automation may be able to provide cost savings (over the long term)
- Manual execution takes too long and automation may speed this up
- Manual execution is too error prone and automation can reduce this, thus increasing the quality of a product
- Execution quality and timing is ad hoc and variable and automation may make this regular and repeatable
It is impractical or prohibitive to use manual methods for example environment builds, database restores, test data set-up
- We have an overall desire to save time, effort and money and we believe automation can help us
- In agile projects which require multiple regressions

Goals of Automation

An organisation which decides to automate may have one, or a number of goals:

- To speed up testing of application releases
- To focus on the reduction of testing costs
- To increase the frequency and the number of regression tests executed against each build
- To increase repeatability and reliability of regression
- To increase the test coverage per test cycle or release

Potential Benefits

There are a number of potential benefits of implementing a robust test automation solution. These include:

- Longer term reduction in the cost of testing
- Accelerated time to market
- Reduced cost of quality
- Increased collaboration with developers
- Shorter feedback loop on quality
- Increased environmental readiness
- Improved consistency of testing processes
- Improved breadth of test coverage
- Increased re-use of tests
- Increased confidence in the quality of testing
- Increased job satisfaction

Uses for Automation

The following is by no means an exhaustive list, since one could add desktop deployment, version control, and release or build management and even LOGON scripts to the list. The focus is on functional automation leading to regression. The following types of automation can be applied to a given project:

- Generation of test data
- Execution of business cases
- Execution of data flows between systems
- Results Analysis, Comparisons

Guiding Principles for Automation

There are a number of guiding principles that will help focus any automation project effectively and help it deliver the right solution:

- Test Strategically
- Test to Mitigate Business Risk
- Test Early and Continuously
- Test Visibly
- Automate for Efficiency
- Test Independently

Success Factors

A good automation test project has the following characteristics:

- Sponsored by the project executive
- Has a valid business case and clear return on investment
- Regularly reports to the project manager
- Has clear, definable risk-based objectives
- Has proper planning and time allocation
- Engaged with the business analysts
- Means something to the end users of the system
- Links with the application developers (external or otherwise) and the system architects
- Is supported by the technical experts of the system
- Has all necessary infrastructure support, e.g. monitoring and diagnostics and administration support, e.g. release managers
- Provides quality metrics
- Integrates effectively with Test Management or Continuous Integration tools and processes

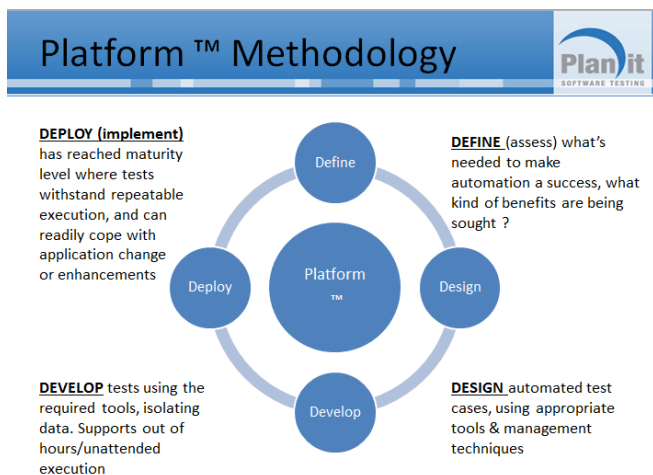
Failure Factors

There are a number of factors that may prevent the successful commencement or implementations of automation:

- No budget/acceptance
- Misunderstandings of the purpose
- Frequent changes or delays of software
- Instability of the target environment
- Lack of relevant automation skills in the organisation
- Limited access to business users
- Unfocused objectives and lack of perceived value
- Lack of continued commitment
- Lack of time
- Inadequate test coverage

Platform 4Ds Framework

The Platform 4 Ds is a framework that sets out the management approach to automation. There is a collateral available which provides support for all phases of the framework such as development standards, templates and frameworks. Platform™ is based on our collective man-years of experience in this field. The collateral exists in the form of a variety of templates and tools to support the ongoing management of the work.



Define

During the Define phase, we set the project up for success, with a viable business case and plan. This stage typically includes the following:

- Defining scope and assessing technical feasibility
- Test tool assessment & selection
- Identifying return-on-Investment
- Creating automation approach and proposed framework
- Deciding on the approach to script development and implementation
- Assessing Remote Execution and Continuous Integration Strategy

Design

Once the Plan is agreed, the Design phase ensures that the correct test automation pack is created. This has the following activities:

- Assessing test scripts
- Test environment preparation

- Test data management and usage definition
- Build and deployment integration
- Reporting and repository management

Develop

During the Develop phase, the automation pack is created. This means the following activities are to be completed:

- Developing test scripts
- Developing test data solution
- Creating recovery scripts
- Ensuring the quality of the development

Deploy

Finally, during the Deploy phase we start to realise the value of the investment made. The following activities are to be completed:

- Executing the test suite
- Integrating into Continuous Integration solution
- Results reporting
- Automation suite defect management
- BAU Handover

Traditional vs Agile Projects

- The 4 Ds framework when taken without further explanation describes automation within a traditional (Waterfall or V-model) project, very effectively. However, for agile projects some or all of the phases and steps are performed iteratively.
- The Define phase is ideally performed once, however the nature of Agile can mean that requirements change and Define must be either redone or adjustments made. This is the entry point.
- The Design, Develop and Deploy phases are performed iteratively. Building test collateral and constantly evolving with the project needs. Test cases are continually developed and executed to ensure they are supporting the project and fully maintained at all times.
- Deployments are smaller and higher frequency, as such supporting tasks for a deployment are performed more regularly. This includes repeated iterations of the Design and Develop phase and occasionally Define as required.

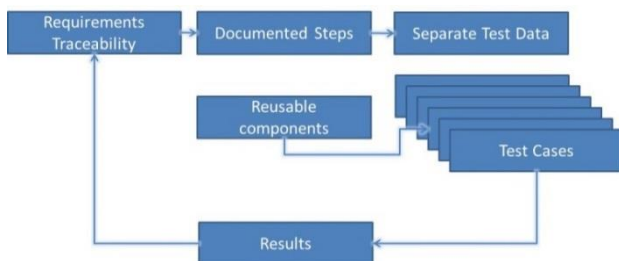
Automation Best Practice

Design

If we look at how software is designed, we see that it is comprised of components which are built into functions and these functions built into systems. In many ways automation projects should follow the same process. The same pre-requisites of sound requirements and good design need to be in place before development starts.

Essential Ingredients

There are a number of prerequisite conditions to enable automation to be applied successfully. These are illustrated below:



Effective automation flows from good analysis and design. Automation should be targeted at specific business needs that are visible and documented. Automation should not be undertaken in an ad hoc or exploratory fashion.

- Good requirements lead to good test cases. These then become viable candidates for automation.
- Poor (or no) requirement or where there is no manual case, makes Automation significantly harder and will likely lead to a poor outcome

The automation engineer has to be able to assist in delivering effective automation, assessing whether an automation test candidate is used infrequently or is of low value and advising on whether it is effective to carry out automation out early in the project lifecycle when the application is volatile.

Script Development Basics

The following is a high level view of which characteristics are ideally required from automation test scripts:

- **Modular** - Scripts should be self-contained
 - Script #4 shouldn't stop script #5 from execution unless they are correlated deliberately
 - Known start and end conditions for consistency
- **Robust** - Additional error handling/anticipation
 - Check we are at suitable 'start' condition and handle that as required
 - Handle errors elegantly
 - To at least return to a clean exit point
- **Well Managed** - Scripts = code = source/version control

- **Extendable** - Suitability to integrate into remote execution environments, suitability to integrate into Continuous Integration and Release tools
- **High Quality** - Needs suitable standards, again for consistency
- **Data Separation** - Need to separate data from the test steps and can be simple XLS, XML, CSV files
- **Well Structured** - Needs dependable preconditions

Automation Framework

An automation Framework is a predefined system that lays the foundations and building blocks required to represent a testing process. They can consist of such things like function libraries, object maps specifying technical properties, and test data specifics. These reusable components are strung together to represent a business process (your test script). The framework can also include a process for test reporting and Continuous Integration.

Automation Framework Types

There are many varying flavours of framework, however all fall into one (or more) of the following categories:

- TDD/BDD capable Cucumber (or equivalent) framework
- Keyword Driven (or Action Word) framework
- Data Driven framework
- Page Object Model framework
- Hybrid framework
- Action framework with object repository

Keyword Driven Framework

The most significant feature of the keyword is the insulation of the business user from the actual automation logic embedded in code. Testing is exposed to business users through a set of easy to understand established keywords that map to generic functions or objects.

The framework maps the actions available to business users, to one or more functions. Scripts can be entirely composed of keywords that are highly independent of the application or UI.

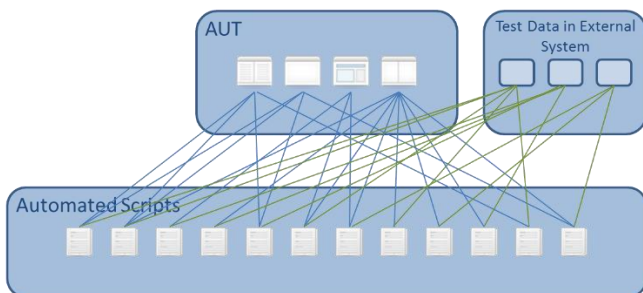
Business users can use the handover information to write their own automation test scripts, utilising their own data and validation logic.

This technique requires a great deal of preparation, however existing keyword frameworks can be more readily adapted to a new application by focusing on the underlying functions themselves, keeping the keywords the same.

Data Driven Framework

This framework approach takes test data from data files which may be .xls, csv, db, xml or json formats for example. This style of approach is commonly used when multiple iterations of the same scenario need to be executed. Each iteration uses varying data from the data files. Validation data (output) can also be obtained from various data files.

The diagram shows the structure of the framework and its interactions with the application under test. As shown the automated scripts access the external data system. This then gets fed back into the scripts and then steers the application under test.

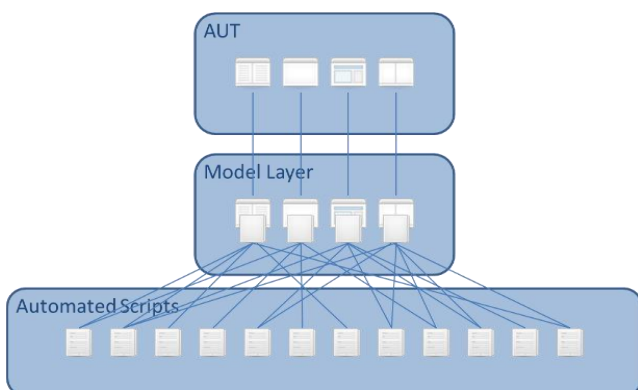


Page Object Model Framework

This style of framework uses page object models to represent sections or pages of the system under test. Technical properties are specified in these independent models and the model is called from the relevant test script requiring it.

A page-model framework supports easier maintainability and usability as the page models represent the application under test.

The structure can be seen below:



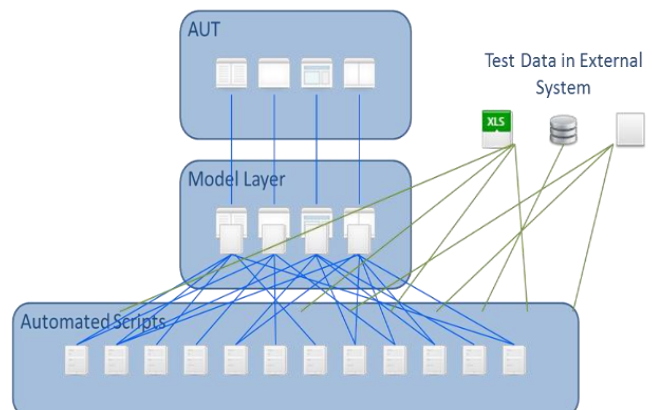
The model layer displayed above holds representative classes containing technical properties and methods that steer or interact with the application under test. Frameworks should also contain necessary reusable utilities and functions that are common across all pages or sections of the application under test. The model maps 1:1 to the application. The model maps 1:n to the test cases. If the application changes you only need to change the model in one place, all of the test cases will be updated accordingly.

Hybrid framework

A Hybrid Framework is a combination of multiple frameworks into one solution – the most common approach seen.

An example:

A Page Object Model Framework is used for the structure of technical properties. This framework can then include a **Data-Driven Framework** approach that allows support for data-driven scenarios while still conforming to the Page Object Model Framework.



Behavioural Driven Development (BDD)

What is BDD?

- A technique to facilitate communication and to define the desired software behaviour
- A common language that can be understood by everyone in the team, whether they have a technical or business background
- An agile practice where the acceptance criteria and the automated scenarios used in both current and future iterations are based on elaborated user stories
- A supplement to provide concrete business examples, not a replacement for requirements

BDD is designed to provide input to automated testing. That is, the BDD's structured scenarios can be parsed by test-generating software to create code to test the application. Automated testing is a powerful addition to the agile development approach. Agile development iterations produce ever larger increments of the end product (release). BDD software tools include Cucumber, SpecFlow and JBehave.

Power of 3



- When the Business Analyst, Tester and Developer discuss and agree on acceptance criteria
- Discussions are upfront and on-going to adapt and innovate
- Supports Agile teams through the focus on collaboration
- Also known as the '3 Amigos'

- Can incorporate a UX (User Experience) role – this is then referred to as the '4 Amigos'

When do the Amigos meet?

- During sprint planning
- When members of the team are in discussions where a decision may impact the user story
- During story development, where the amigos periodically check in together to ensure the story is on-track
- To verify acceptance criteria have been met – this quality gate can be added to the definition of 'done'

Writing Acceptance Criteria

Uses the Gherkin notation of "Given, When, Then"

Given – shows any prerequisites of the scenario

When – actions to be performed to reach the test objective

Then – post verification action to validate the outcome

Why BDD?

By incorporating BDD and the associated principles like the Power of 3 into the sprint planning and day-to-day agile execution, it promotes early collaboration and feedback. This works hand in hand with ensuring everyone is 'on the same page'.

Using a natural language like Gherkin, all included parties can understand the desired behaviour of the software – this helps throughout the sprint to bridge the gap between technology and business.

The collaborative definition of acceptance criteria and the shared understanding promotes and aids accuracy in story points/estimates. While it may be a change to the normal way of writing test scenarios, implementing BDD provides the team with a single view of acceptance criteria. As the testable scenarios are also the acceptance criteria, the usual test case design layer does not need to be repeated, meaning there is less risk of incorrect translation from requirements to test cases. Early formulation of test scenarios also allows early identification of requirement gaps.



Automation Technical Concepts

The choice of where and how to automate used to be relatively straightforward, but not so in today's distributed and highly disparate environments. So we need to consider the following factors like:

- Could automation be done at a screen (UI) or interface level?
- Is the system entirely self-contained or is it dependent upon third-parties and external components?
- Is the system new, an upgrade or a mix of the two?
- Does the automation support both web-based and digital systems?
- Do the staff have the expertise to manage automation internally?

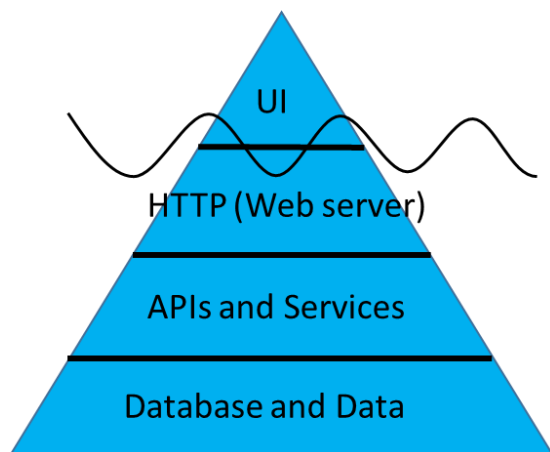
Answers to these questions help to drive us towards the correct automation solution and answer the questions of where and how.

Tip of the iceberg

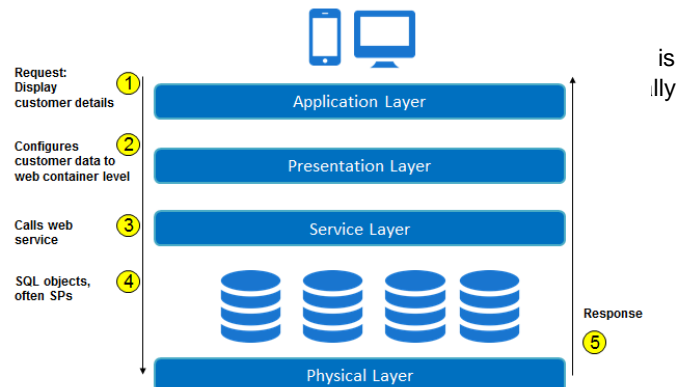
The iceberg analogy is intended to convey to business users the hidden complexity in the application. This helps discussion on the amount of work required to build an environment or release that has all the appropriate components within it.

It also facilitates the argument that functional regression goes beyond what the screen may display.

Think of your current system as an iceberg, 9/10ths of it is invisible or behind the screen.



In older architectures, system boundaries were relatively clear and the user interaction was relatively simple. In modern systems architecture, there are systems with multiple interconnections, 3rd party links, multiple integrated systems, and often SaaS (cloud) solutions. Screen interactions and accesses are more complex with many touch points, external and 'rich' user Interfaces.



User interface (UI) based automation

Driving the user interface is the most common method of test automation, found in tools like UFT, Selenium, Tosca and many others. These tools interact with screens, buttons, text boxes and mouse clicks.

The major drawback of this is the time spent to understand, deliver and execute a UI solution and the lack of certainty that the interface will be supported out of the box.

Experience has shown that just because an app is web-based doesn't mean automation will be simple. The only sure way to know that a given tool supports your application is to trial it (perform a proof-of-concept). One of the key concepts that underpins successful automation is to understand technically how the tools works with a particular combination of technologies.

This means that UI automation tools have to be able to intercept and interpret actions such as open a window, click on a button, or enter text. Visually this may appear simple, but exactly how that is done varies hugely under the covers.

Pros	Comments
It's the most intuitive to understand	So it's understood by stakeholders or non-technical audience
It's the most complete end to end approach	So we are exercising all downstream components in our tests as well
Cons	Comments
It can be unpredictable/unreliable	Error handling, warm restart
Vulnerable to UI changes	Although object maps and frameworks reduce some pain
Most costly to execute time wise	

Test Automation Fundamentals

Business Logic Layer Automation

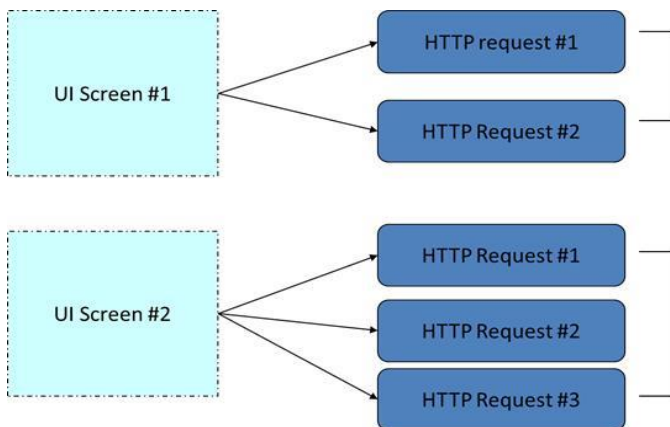
In a project where the screens have yet to be fully developed it may be better to look at automating 'behind the scenes' at the interface level. If the screens are a work in progress and liable to change, some level of UI automation re-development would be inevitable.

HTTP (Web Service) Automation

If the application is web-based, automation can also hook into the underlying HTTP messages that flow between the client (web browser) and the back-end (typically a HTTP web server) before progressing onwards to messaging or service layers.

This type of automation is common particularly in performance testing. This is because it is the most reliable and scalable way of simulating hundreds of different users.

In a web application a single screen may trigger one or more HTTP requests, which need to be simulated and parameterised



At first sight it may seem irrelevant for functional testing, but in some circumstances HTTP can be used to drive regression, especially if we focus on services.

Pros	Comments
Don't need to wait for the GUI to be ready	We can validate at the service level
Quicker to execute than normal GUI automation	
Cons	Comments
Large amounts of correlation may be required	A service may require another service to execute e.g. Authorisation

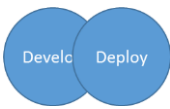
--	--

Automation of APIs and Services

Automation has driven deeper down the technology stack, and moved away from the delivery of the User Interface (UI). Service-based automation is a good candidate for automation because of the standardisation of web services based on HTTP / SOAP, and protocols like REST, MQ, JMS etc.

On some projects the initial development effort may concentrate on system-to-system interfaces ahead of the user interface. Indeed the project may need to support multiple interfaces (browsers, both PC and mobile). There has also been a significant rise in Service Virtualisation tools like IBM Green Hat or CA Lisa, which stub or simulate parts of the interface e.g. System A talks to System B (which hasn't been built yet). The tools stubs 'take the place' of System B until it is ready.

Pros	Comments
There is a common standard as systems become more 'stretched'	Relevant to today's disparate architectures. Services are self-describing
It breaks away from the pure screen approach, and is less cumbersome than HTTP	Avoids the inherent problems in testing at screen or sub-screen HTTP. It's a clean approach
It's now supported by test tools such as SoapUI, CA LISA, IBM Green Hat, HP UFT	Wide-ranging support for web services, and allows us to stub or mock when systems are limited or unavailable
Cons	Comments
It can be cumbersome to program	If realistic business cases are needed
Not end-to-end in its entirety	More suited to system-to-system tests
It's less intuitive for business folks	Works best at system or integration phase



Automation Tool Capabilities

Test Automation requires a test tool! There are a large number of tools available on the market. These tools differ; by look and feel, by capability, by the scope of what they can automate and by how they record an automated script. All tools have a similar base set of capabilities; what differs is how they deliver these capabilities.

Creating new projects

Each tool has a different way of storing, categorising and displaying how the tests are organised. Most of this is generated by the tool. However this is done, the first step in using a tool is normally to create a project.

Record tests using record-replay

Record-replay (or capture-replay) is the most basic form of test automation. It is simple to use but is very limited in what it can achieve. This is a simple activity of starting the record process, manually performing the operations you want to record and then stopping the process. Some tools are only record-replay but conversely some tools do not have this capability, instead having a requirement for script development or keyword development. For those tools that do record-replay this is not generally regarded as good practice and typically used only a start to the automation process by capturing flow.

Once you have captured some activity, you need to understand what you have created. A record-replay activity may create the following automation project components:

- **Script** - end to end representation of what is recorded and what will be played back
- **Screen objects** – what the application has identified as the objects it needs to interact with
- **Actions** – keyword or script language components describing what the script is doing

Amending record and replay tests using script or keyword languages

For those tools, which allow this, an automated test using record-replay can be progressed by directly amending the keyword script or language script that is created. It is this script that allows loops, logic and validation to be added and a single test to become a suite of tests. It also forms the basis of modularity (see later). Key considerations include:

Keyword scripting tends to be proprietorial, differing by tool and easier to understand than scripting. Keywords allow logic to be incorporated and data to be referenced

Scripting is real code and needs real coding knowledge. It provides the advantage of being able to use all the capabilities of code and comes with all the complexities. A good automator needs a good understanding of coding best practice, methodologies such as object oriented design, and coding standards.

For tools which rely on scripting languages, many do not have a code interface, instead relying on integration with Integrated Development Environments (IDEs) such as Visual Studio (for Microsoft Technology and Eclipse for Java technology).

Record tests using scripting, BDD or a keyword language

More experienced test automators or developers will go straight into a scripted automation solution using the tools script interface or using an IDE and Unit Test framework that the tool integrates with. This has a number of advantages:

- It is quicker to implement than the process of recording a script, checking the recorded components, adding logic, and finally testing the reworked script. An experienced test automation specialist can write code quickly and be able to re-use code from previous activities.
- An off the shelf Unit Test framework provides the ability to integrate with the developer's unit test effort, easy test execution and reporting. Additionally it offers integration interfaces with test management, CI and monitoring tools. Being off the shelf it significantly accelerates framework development.
- Re-use of code lends itself to more structured and modular approaches to automation. Using coding standards and best practices, a test automation solution can be created with as high a quality as the code it is intended to test. This is why test automation practitioners can be called "developers for test".

There are some considerations:

- Large frameworks require significant and advanced programming skills. It can be hard to find this within the skillset of a typical testing team.
- Like any development effort defects will be introduced during the development process. This can be frustrating and inefficient if a proper development lifecycle is not being followed.

Results Validation

All tools provide a capability to check whether the results of the tests are as expected. This comparison can be on the screen, database, on a file, or indeed anything that can be checked. The following considerations should be taken into account:

- Not everything can be or should be checked. Only compare the most important results. Manual checking will still be needed in most test projects.
- Validation can be dynamic (during execution) or post execution
- We need to understand what to do when there is a deviation. This is known as error handling and must be designed into the solution.
- Thought should be put into how results are reported



Test Automation Fundamentals

Modularity

Modularity is the process of taking often re-used elements such as login information and creating a separate script. All tools allow modularity through their interfaces or via IDEs. Modularity:

- Needs to be planned in advance
- Increases re-usability and speed of development
- Reduces maintenance as amendments need to be made in one place only
- Reduces complexity of scripts and increases clarity

Parameterisation and data input

Parameterisation allows the same cases to be run with different data. It is good practice to separate test flow steps from actual test. This allows re-use which speeds up the test development effort. Some tools have their own interfaces to store data, others can point to spreadsheets or external databases. Open source tools usually implement a Cucumber or equivalent framework. This uses Gherkin to specify the test cases in natural language and allows for them to be stored in feature files that contain scenarios. These can then be parameterised.

Some things need to be considered:

- Different data will often drive different flows in an application. Data is therefore intrinsically linked with the input of logic into the script.
- Test data needs to be well managed to ensure it remains fit for purpose. (this is also true for manual testing)

Error handling

Error handling is:

- Handling failed test cases, where an actual result does not match the expected result
- Handling script failure, where the automation pack no longer functions correctly

When planning error handling you need to understand how robust you want the script to be and what to do in the event of each error (log and continue, wait and try again or stop).

Addition of logic

The addition of logic to automation scripts is needed in all but the most simple automation projects as application flow is not linear. Different screens or functions are visible dependent on choices made by the user. Automation needs to cater for this. This is typically done through keyword or script amendment, though tools with codeless interfaces also allow logic through hierarchical or tree notations.

Synchronisation

Synchronisation is ensuring that the automation script can handle the time taken for the application to perform. If the script runs too fast, then it will be trying to perform actions on objects or functions that are not available. Too slow and some of the benefit of automation is not realised.

Synchronisation can be performed in a variety of ways:

- Inserting explicit delay e.g. number of seconds

- Inserting commands which wait until something has happened

Scheduling and Execution of tests

Preparation before running a test automation pack is important. We don't always want to run a full pack. Choice of how much to execute is based on the objectives of the run and how much change there has been. A smoke test for example will only test a small subset to ensure the environment and application is capable of supporting further manual testing.

Before execution we often need to set up the run; this will include things such as pointing to the correct environment and data selection.

Some tools allow time scheduling so that tests can be run out of hours and distributed amongst multiple machines. Dependent on the robustness of the suite, it may be better to monitor the run, rather than leave it unattended.

Automation can be scheduled as part of Continuous Integration, so the timing of execution will be determined by this framework. In practical terms, this means that execution will be triggered from not just the test tool itself but from the IDE, the CI tool or from other integrated test management tools.

Reporting

Tools often vary in the amount of reporting functionality provided. Reporting is usually provided at the end of execution, though it is possible to have reports generated on the fly during test execution.

Different reporting levels are required depending on the stakeholders the report is meant for. For example an Automation engineer may require detailed debug information to do root cause analysis on a test failure. A Test Manager will only need high level information such as the test cases executed and the pass and fail rate.

Tools typically provide the following reporting functionality:

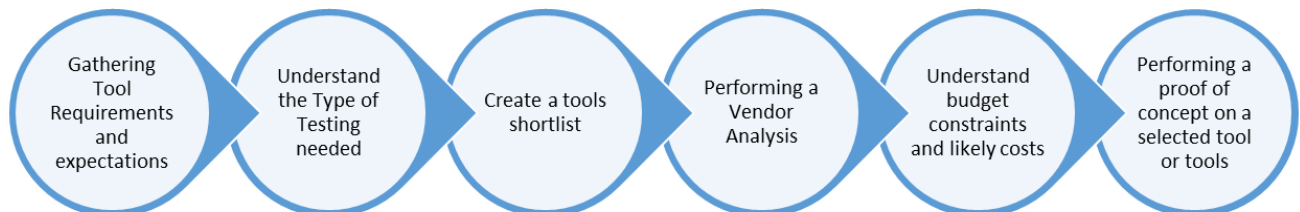
- General stats such as which and how many tests were run, as well as and how long each test took
- Reports on validations, how many tests passed or failed and what the discrepancies were. Many tools can be configured to provide screenshots for each step, or on failure
- Which test environment was used
- Graphing or visual representation of the run
- Some report into other tools such as the IDE or test management tools
- Some can be configured to directly email reports at the end of the run

Unique features

Each tool has unique features that are marketed as improving user experience, effectiveness or efficiency. In some cases these are truly unique to the tool, in others variations are of one of the capabilities detailed above.

Define Tool Evaluation and Selection

The choice of tool greatly impacts not only the development of automated tests but also the engineering process within an organisation. In order to select an appropriate tool the following steps will be taken.



Automation Tool Requirement

A good way to gather information about tool expectation is to meet with analysts, architects, environment specialists, developers and try getting answers to the below questions:

- Is there already a tool?
- How will the organisation use the tool?
- Is there a budget?
- Who will be the main owners of the tool?
- What protocols and technologies does the tool need to support?
- Do technical or non-technical users use the tool?
- What are the core features the tool must have?
- Does the tool fit with the overall development methodology within the organisation?, e.g. Agile or Waterfall

The automation engineer should assess the systems and software architecture that are commonly used across projects and environments within the organisation and also should consider the following factors:

- Other projects may use different technologies and may have a current or future need for automation
- Automation tool selection may need to support multiple message formats for different application types
- The methodology used for development within the organisation
- What stage of testing is Automation being introduced at? E.g. at Requirements, development or after the manual testing phase is completed

Ideally, an automation tool should not be selected for just one project, but rather cater for significant projects and applications within the organisation and be a strategic decision.

Assessing the Viability of Automation

Process Viability

- The test process is clearly defined and facilitates the use of existing test artefacts

- Test automation requirements are clearly defined
- Test automation has a defined architectural approach and supporting design
- A robust implementation that provides appropriate levels of reliability
- Repeatable automated tests
- Minimisation of software change impacts to the automated tests
- Clearly understood and documented levels of interdependency
- Test automation standards are defined and followed
- Test automation effort must be planned, documented, monitored and measured
- Clear definition of roles and responsibilities for use of the developed framework and its ongoing support and maintenance
- A champion has been clearly identified within the organisation to perform a cultural change management role to facilitate adoption and use of the framework within the organisation
- Test environment is controlled to ensure a high level of stability which enables repeatability

Technical Viability

Some technologies or environments do not extend to automation i.e. There are technical limitations of the automation tool interacting with the current technology stack or environment required to be tested.

Type of Testing needed?

The types of tests needed by the project will greatly determine the selection of the tool. Types of tests may include:

- Unit testing
- Component/System/Integration - Regression testing
- Usability testing

The **Test Automation Pyramid** (on the next page) gives a visual representation on where test automation is mainly focussed. The widest coverage is with the unit tests and then the acceptance tests, with most exploratory testing being manual. It should be noted that there are some tests where it

Test Automation Fundamentals

will never be economic to automate, to understand this all tests should go through an analysis phase before they are automated.

Tool Shortlisting

A Shortlist of tools should be created that matches the requirements gathered. In addition the tool should be considered as a long-term investment by the organisation that can be used for a number of different projects and departments. As a result, the automation engineer should consider not just the project requirements but the long term greater organisation's requirements. The shortlist should factor this in.

The shortlist will typically describe 2 or 3 suitable tools and the differences between them. The Pros and the Cons can be described and it may be that a likely recommended tool emerges.

Overview	BDD support
Pros	Continuous Integration support
Cons	Test Management tool integration
Manual Testing Capabilities	Supported Technologies
Test Management and Reporting	Supported Browsers
Traceability of Requirements	Cost
Defect Management support	Mobile Testing Capabilities
Optimised Test Case Design	Codeless User Experience or not
Data-driven Testing Capability	Programming Languages Supported

Vendor Analysis

Other factors should be taken into account with regards to who owns the tool as not all vendors are created equal.

- Who are the vendors?
- Where are they based?
- What is the development roadmap for the tool?
- What support, tutorials, training do they provide?
- Is there an online community

- How reliable is the tool?
- What is the current version?
- Are there regular updates?

Budget Constraints

What is the testing budget and how much has been allocated to the purchase of an automation tool? This information is critical in the decision making process, as automation tools have a wide range of costs, some being free and open-source and others being commercial with costs related to licenses and support. The test engineer should evaluate the following cost factors:

- Cost of tool purchase/licenses
- Cost of training or recruiting automation testers
- Cost of support
- Maintenance fees (subscriptions)
- Add-ons/plugin-ins
- Future scalability requirements

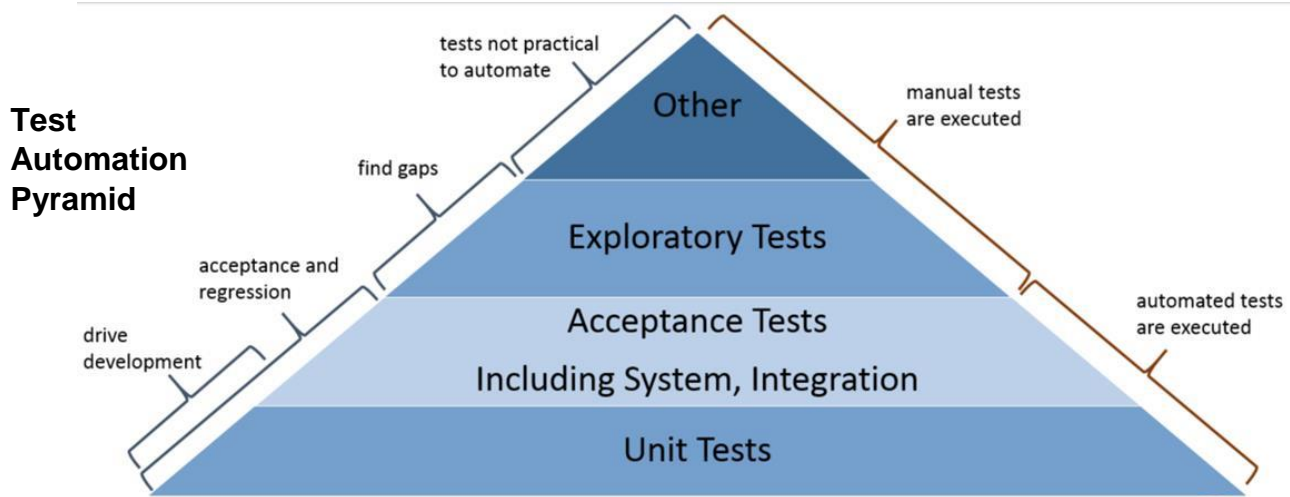
Proof of Concept

A Proof of Concept (POC) is a focused evaluation exercise with the end goal of being able to conclude the validity of a tool or concept prior to a streamline implementation. It is a highly recommended preliminary activity when selecting a new tool and answers the following questions:

- Have we confirmed that it meets our Key Business requirement?
- What amendments or support are needed to meet our technical requirements?
- Can the tool be implemented successfully in the target organisation?

A POC typically takes up to 3 - 5 days for each tool and follows the following steps:

- Implement the tool onto the test environment
- Select a valid business process or test script to automate
- Record the script in the automation tool
- Execute the script
- Produce a final report and recommendation



Design

Automation Tool Examples

The automation tools market is very diverse and crowded. In this section of the guide some of the leading contenders are described, identifying relative strengths and weaknesses and other factors which differentiate the tools.

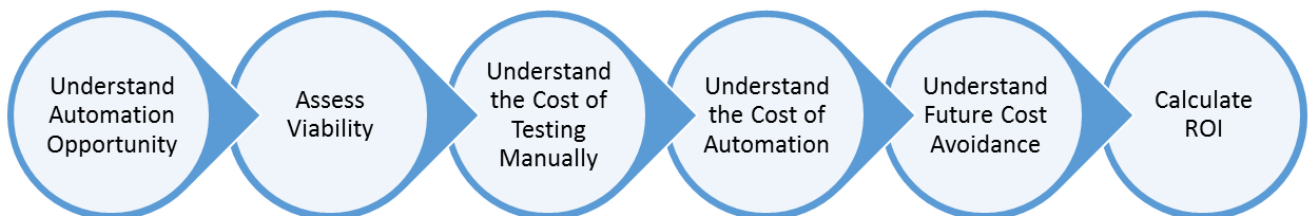
Popular Tools

<p>Selenium</p> 	<p>Selenium WebDriver is an open source tool used for automating websites and supports a number of different programming languages. It requires an Integrated Development Environment such as Microsoft Visual Studio when using C# or Eclipse when using Java. It is usually used in conjunction with a Unit Testing Framework. Selenium IDE is a Firefox browser plugin that adds record-replay functionality to Selenium. Selenium has a very strong market presence due to the speed of its updates, support for most languages and absence of initial outlay. Selenium can also be integrated with API/DB/Mobile testing drivers to deliver cross functional testing capability. However, support is only provided through an open source community and not via a vendor.</p>
<p>HP UFT</p> 	<p>Formerly Quick Test Professional, HP integrated API testing capability and renamed the new offering HP Unified Functional Testing (UFT). It supports both record-replay and scripting approaches (using VBScript). UFT supports a wide range of technologies in addition to APIs, these include but are not limited to web, SAP, Siebel, native windows, .Net, Java and Terminal Emulators. In addition, it integrates with the HP Application Lifecycle Management (ALM) Test Management toolset. HP UFT has mobile technology capability with HP Mobile Centre.</p>
<p>SmartBear Test Complete</p> 	<p>Test Complete is primarily a scripting based automation tool with record and playback functionality. It can be implemented in multiple languages. It supports a range of technologies including web and desktop applications, and offers mobile testing capability. It has two key additional useful functions; a visualiser which enables the tester to add steps to an existing script by navigating through a previous recording, and a data generation facility which creates data using standard popular formats such as e-mail addresses and post codes.</p>
<p>Tricentis Tosca</p> 	<p>Tricentis Tosca Testsuite defines itself as a third-generation technology using a model-based approach; differentiating itself from record-replay tools or tools that use code-based technology. Tosca has “Dynamic Steering”, which allows the design and specification of manual and automated test cases in a purely business-based way. The technical interface to the application, the models, are called modules. These are then linked to the test cases. This allows the test cases to be maintainable by testers that do not have technical knowledge. Test case design is supported by Tosca by means of combinatorics-based test creation of mapped flows. Tosca supports a wide range of technologies including SAP and also now has elements of Test Management designed into the suite, as well as mobile testing capability.</p>
<p>Ready! API</p> 	<p>Ready! API is a suite of tools ranging from security and performance (Secure Pro & LoadUI NG Pro) to service virtualisation and automation (ServiceV Pro & SoapUI NG Pro). SoapUI NG Pro in particular provides the capability to test API protocols such as SOAP and REST. The tool’s point-and-click capabilities allow a user to incorporate the platform’s functionality into their testing without the need for complex scripting. SoapUI NG Pro supports integration with unit testing frameworks such as JUnit, and build automation tools including Apache Maven.</p>
<p>Ranorex</p> 	<p>Ranorex is a test automation tool that supports a wide range of technologies across desktop, web and mobile. Ranorex has script-free testing for non-programmers and a professional API for C# and VB.NET. It also integrates with Visual Studio. Ranorex supports cross-browser testing on common browsers and can handle common web technologies. It supports mobile operating systems like Android & iOS. It helps test mobile applications on real devices and emulators when they are connected to a PC through USB or Wi-Fi and eliminates the need to Jailbreak mobile devices. It also supports cross platform execution of tests written for websites on desktop to run on mobile browsers. It can be integrated into any continuous delivery process and a test suite can be triggered from any continuous integration systems or any test management tool.</p>

<p>MicroFocus Silk Test</p> 	<p>MicroFocus Silk Test is marketed at teams with multiple skills sets, with a codeless interface as well for less technical team members, and IDE integration for more technical staff to use. The latest version of Silk Test incorporates, mobile testing technology into its platform, rather than having separate applications for desktop, browser and mobile testing. It relies heavily on its device lab functionality to manage multiple devices with the same test. It provides a range of tools for continuous integration. Silk Test provides support for a large number of platforms and applications, as well as the ability to use C#, Java and VB.Net. Silk Test has its own scripting language, 4test.</p>
<p>Cucumber</p> 	<p>Cucumber is an open source tool which supports automation of acceptance test cases for web applications as part of Behavioural Driven Development (See Appendix b). Cucumber functions as a testing framework, rather than a tool itself. Cucumber uses a Business-focussed syntax called Gherkin to define tests which allows project team members (Business Analysts, Developers Testers) to use an easily understandable format to communicate. Cucumber integrates with Unit Test Frameworks and automation tools and can use programming languages to automate testing direct from the Cucumber Interface. Typical deployments use Cucumber with Selenium or Cucumber with Java.</p>
<p>SpecFlow</p> 	<p>SpecFlow calls itself – Cucumber for dot net (Microsoft technologies). It too supports automation of acceptance test cases using Gherkin Syntax in order to support BDD. It uses Microsoft Visual Studio as an IDE and can therefore use Visual Studio functions such as debugger support. The output of a project is usually an MSTest or NUnit test assembly. This provides provision for integration with CI or CD (Continuous Integration and Continuous Development) solutions.</p>
<p>MicroFocus Silk Mobile</p> 	<p>Silk mobile is a mobile automation tool for testing native apps and mobile websites with record and playback functionality. It provides functionality to write code-free programming. It supports several mobile operating systems like Android, iOS, Symbian, Blackberry & windows mobile. It helps test mobile applications on real devices & emulators when they are connected to a PC through USB or Wi-Fi and eliminates the need to jailbreak mobile devices. Silk Mobile supports Native object recognition, Dynamic Object recognition and Optical Character recognition and also provides features to run tests on multiple devices in parallel thereby decreasing the amount of test time and speeding up the release process. It also provides functionality to implement tests across a variety of test scripting programs like Silk4J, Silk4Net, JUnit, NUnit, C#, MSTest, VB.Script, UFT (QTP), Ruby, Python or Perl and be a part of the continuous delivery process.</p>
<p>Appium</p> 	<p>Appium is an open source mobile test automation framework for use with native, hybrid and mobile web apps. It drives iOS and Android apps using the WebDriver protocol. It allows you to run the automated tests on actual devices, emulators and simulators when they are connected to a PC through USB or Wi-Fi. Its API can be extended to allow users to write & execute tests for wearable devices like Apple & Android watches. Appium provides functionality to implement tests in a variety of programming languages like C#, Java, Python, PHP & JavaScript. Appium doesn't dictate the test automation framework to be used and can be used with any test automation framework. The Appium clients are simple extensions to the WebDriver clients and most of the Selenium WebDriver commands are available to Appium.</p>

Define Return on Investment

Understanding the likely return on investment from providing a test automation solution is necessary to develop the business case that will help provide funds for the automation project. This will be done by understanding some of the technical and business factors that influence the environment and technology.



Understand the Automation Opportunity

In order to successfully implement an automated testing solution, there will need to be technical specialists, as this is a task that the business representatives will not have the skills or the time for. Additionally, consider the software development maturity level of the organisation; it may not be feasible to start the creation of a large scale test automation framework if the organisation is not mature enough to support it. In this case it may be more suitable to suggest a simpler solution such as a record and replay pack rather than a full automation framework (though this is by no means always an effective or recommendable solution). Also, we need to consider why automation is wanted and whether this is justifiable, for example whether it is for a quick win or for a long term benefit? Finally, we will need to consider what tools will be used for automation and the costs of these.

This leads to a list of factors that will provide input into the building of a business case for automation. There are:

- The business drivers for automation
- The objectives and whether these are realistic
- Application stability and consequent likely repeatability
- Test Environment costs considering in-house, virtual or cloud options
- Tools, options include 'open source' vs commercial
- Current Manual and automation solutions and how much cost is associated with these
- Any Current investments which may become obsolete
- Cost for both Initial acquisition and ongoing maintenance
- Skills needed to implement and other Technical dependencies
- Whether there is easy access to business users and other functional SMEs
- How much will automation be used, taking into account the frequency and duration of cycles, releases and patches?

- Are there any costs associated with training or upskilling current staff? Do additional resources need to be brought in?
- **Lifetime of Application Under Test** – Will the application be redundant soon

Automation of 100% test coverage may not be the best use of time and money; Business value, Risk, Feasibility and Viability all need to be taken into consideration when deciding what to automate.

Focus efforts on automating areas or functions that provide a higher **Business value** to the business and/or areas of the application frequently used by customers.

The level of **risk** of the application under test should be considered concentrating on understanding areas of the application which are frequently used and/or where a high number of errors are seen.

Assessing the Viability of Automation

Process Viability

- The test process is clearly defined and facilitates the use of existing test artefacts
- Test automation requirements are clearly defined
- Test automation has a defined architectural approach and supporting design
- A robust implementation that provides appropriate levels of reliability
- Repeatable automated tests
- Minimisation of software change impacts to the automated tests
- Clearly understood and documented levels of interdependency
- Test automation standards are defined and followed
- Test automation effort must be planned, documented, monitored and measured



Test Automation Fundamentals

- Clear definition of roles and responsibilities for use of the developed framework and its ongoing support and maintenance
- A champion has been clearly identified within the organisation to perform a cultural change management role to facilitate adoption and use of the framework within the organisation
- Test environment is controlled to ensure a high level of stability which enables repeatability

Technical Viability

Some technologies or environments do not extend to automation i.e. There are technical limitations of the automation tool interacting with the current technology stack or environment required to be tested.

Assessing Manual Costs

At this point, we consider the cost of continuing with ongoing manual testing against the implementation of the automated execution. It is important to set costs realistically and also to set expectation levels carefully.

Example: Calculating Budget Savings Baseline Values

- Average resource rate is \$480 per day or \$60 hour (8 hour day)
- A manual test case takes an average of 2 hours to complete
- Therefore, costs \$120 (i.e. \$60/hour times 2 hours)
- 4 x manual test cases per day (i.e. 8 hours divided 2 hours)

Assessing Automation Costs

Initial costs include tool acquisition, training, and script development or framework enhancements. As systems change over time, a % factor should be applied to assume that some of the core sets of test assets will incur a maintenance cost too.

One thing to remember with budget saving is that not everything can be automated. Current figures suggest that typical automation coverage of 20-30% of a regression set can be automated, although some tools claim to have up to 60%.

Future Cost Avoidance

- **Positive ROI** – Cost of automating the test case versus manually running the test case while also taking into consideration the cycle/iterations this test case may go through during the test execution phase.

Cost avoidance looks at areas where automation saves on delays, downtime or costly post-production fixes.

- Regular Environment Readiness testing

- Before each release or major configuration change
- As new systems come on board
- Continuous Integration benefit; reducing release and configuration time.
 - Continuous testing provides a heartbeat for the environment, this includes uptime and basic regression functionality. Tests are executed each time code is committed
 - Defects are found earlier, often before they are even deployed in a test environment. This reduces defect triage and fix times, reducing cost.
- Savings of 'nn' man-days in equivalent manual testing
 - Further savings as this effort can be done out of hours, helping keep critical path on track
- Savings NOT only test execution time
 - There are also potential man-days saved as a result of error detection.
 - i.e. we avoid time fixing or investigating \$\$ expensive errors
- How well can a manual tester verify data content?
 - They can't, especially not vs. automation tools!
 - Particularly relevant to interface (SOA or ESB) type testing
 - E.g. payments hub, data interchange systems

Return on Investment

Future savings looks at analysing the break-even point where automation set-up has been paid back, and hence the benefits from automation are self-funding.

- At the end of the automation project
 - an automated test pack will be created
- Each time this test pack is executed
 - will save in the region of 'nn' hours of manual test execution
- Further savings can be realised on the 'scripting time' for subsequent projects
 - they don't need to script the unchanged components of the test pack
 - Staff skills and experience has increased, reducing script development time
- So as the programme or project matures (medium to long term), savings in existing investments accrue
 - So they become more 'profitable' (or have increased ROI)

However, do not forget that there will still have to be some ongoing maintenance.

Design Data Generation

Automation is a powerful tool for data generation. There are a number of ways in which test automation can address this requirement. These are reviewed and the strengths / weaknesses of them considered.

Options & Methods

- Option #1 - Drive the User Interface or Application
 - e.g. a screen based automation tool
- Option #2 - Drive application/HTTP interface
 - e.g. a sub-screen based tool
- Option #3 – Drive system-to-system interfaces
 - e.g. tools that support messaging/service calls
- Option #4 – Specific database tools
 - e.g. commercial tools designed for databases
- Option #6 – Your own test harnesses
 - e.g. customised programs

The variety of options available is good – plenty of flexibility in the choice(s) to be made. The table below illustrates options with examples of the kind of tools that may be used to automate our data.

Option	Example	Comments
Test Data Generation <i>At screen level</i>	Selenium, UFT, Tricentis TOSCA	Scripted driving of the User Interface
Test Execution (en masse) <i>At browser level?</i>	SoapUI, Green Hat, UFT, LoadRunner	Scripted driving of the HTTP requests/responses
Test Execution (en masse) <i>At interface level</i>	SoapUI, Green Hat, UFT, LoadRunner	Scripted driving of the messages, MQ, Soap
Data Generation <i>At table/view level</i>	SQL Data Generator	Flexible generation at database with Ref integrity
Programming	Ruby, PHP, VB, Java, C#, many language options	Custom harnesses for specific data tasks

Screen or UI based generation

The most obvious way to generate data is through the application itself. Data injection via the UI has some merits, provided the constraints and requirements are clearly understood. UI generation can be used for simple data requirements in unit testing or simple system testing environments. One of the advantages is that anything behind

the scenes such as third-party systems is handled automatically.

Following are the some of the Pros and Cons of screen based generation.

PROs	Comments
No messy correlation or end to end integrity issues	We automatically benefit from application integration
Can reuse existing automation scripts	But dependent upon state of development
CONs	Comments
Screens have to be available and working functionally first	Can't start ahead of UI development
Only goes as fast as the screens	Would not generate thousands of entries as quickly as other methods

Performance tool data generation

If thousands of data entries are required rather than tens or hundreds, then driving the UI may be too slow. A performance test tool such as LoadRunner can overcome scalability issues since it is designed to run many instances at the same time. Creating 1,000 new accounts via a Performance test tool may take only a fraction of the time that a Test Automation tool like UFT would take. Following are the Pros and Cons of using performance tools in data generation:

PROs	Comments
Faster and more scalable than a UI based approach	Scripts are intended to execute at volume, speeding up data creation
Reuses existing scripts just like QTP et al	But dependent upon state of development (especially hidden fields)
CONs	Comments
Messy correlation or end to end integrity issues?	Makes programming of the scripts less intuitive and more specialised
Screens or interfaces also have to exist first!!	Still constrained by the state of the application

Test Automation Fundamentals

Interface generation (SOA)

Applications today have a complex 'n-tier' architecture – the advent of standardised interfaces such as JMS (Java Messaging Systems), or HTTP/SOAP and web services makes it possible to hook into these interfaces and harness them for data generation.

A SOAP interface is well documented - indeed it is self-describing. This means a web service will publish how to send it a request and what the response looks like. The other side of the contract is that the web service states clearly what data is returned from this call.

Tools such as HP UFT, SoapUI or IBM Green Hat provide a natural fit with this service-oriented model. Alternatives are plentiful, including CA Lisa and Tosca.

SQL Data Generation

Another data generation option that works at a lower level than the application or interface it uses, is the database or specific database generation tools. Again there are too many options in the marketplace to provide a comprehensive coverage in this course. Examples are:

- Microsoft Data Generator available through Visual Studio
- A range of Redgate tools which provide SQL Comparison at schema (database design level) and content (database values) level
- SQL Tool belt, which provides Backup / Restore / Refactoring capabilities
- SQL Data Generation and also Data Cleanser
- Cloud tools such as database admin for Azure or EC2 instances

Pros and Cons of SQL Data generation are:

PROs	Comments
It's very fast to execute compared to UI approach	Generates thousands of rows of related data in seconds
SQL tools are very cost effective	Budget friendly
Extensible and customisable	Built in 'attributes' easily change plus add new attributes quickly
CONs	Comments
You really need to know and understand the data model	Can easily insert 'duff data' into your database which may not be relevant or work from your UI/Front end so your tests may fail due to quality of data
Not designed for end users?	How to code regular expressions or more complex Referential Integrity rules
Tools mainly SQL Server based	Can be hard to convince stakeholders of the ROI of using these tools
This works only on a single source database	Different approach would be required on a GDW (General Data Warehouse) or complex SOA Enterprises

Custom Programming

The use of popular coding languages like Java, C# or Ruby can also provide a harness to generate data. The power and open-source-like extensibility are two significant factors. Pros and Cons of scripting languages are:

PROs	Comments
Very fast to execute compared to UI approach	Generates 1000s of rows of XML data in seconds
Open-source/widely known	Low budget, skills required to pick up
Extensible and customisable	Huge community of add-ons/plugin-ins
Lots of potential uses	Supported by the sheer variety of add-ons
CONs	Comments
Not as instantly data friendly as SQL Generator	Could be used to access database tables and produce simple extracts
Not designed for end users!	It's for coders/developers
It's open source	May clash with in-house culture and existing investments



Automation Maturity Matrix

This section of the course looks at progress through stages in test automation, starting from basic record/replay to parameterisation through to keyword driven frameworks.

5 Maturity Levels

There are 5 categories of automation explored, although basic record/replay is not considered mature. There must be at least an element of reuse and re-execution to make it to what is considered 'managed'.

The interaction and inter-dependence between business users and technicians who support automation is highlighted, as automation changes from simpler parameterisation to hybrid frameworks.

As the automation techniques improve and become more object-oriented, the coupling between the tool of choice and the application becomes more flexible, allowing users more autonomy in the testing process.

Automation maturity is addressing the Develop & Deploy stages of the Platform™ Methodology

Level 1: Initial

At this level, automation has not been established as a testing capability and the reliance is on manual testing.

Moving upwards requires:

- Capability uplift through training/or skilled resources
- Initiating a POC phase and assessment of suitable tools
- Piloting an automation project is also a good idea to ensure it can align to your current software development environment and processes

Level 2: Managed

Automation is deemed out-of-the-box if scripts are simply record-and-playback. There are no customisations of scripts and scripts may be implemented after manual testing therefore applying a level of duplication to the effort. Limited coverage is attained due to being 'tightly coupled' to the UI. If the screen changes, rework is almost certainly required. There is also lots of technical hand-holding as no synchronisation or wait customisations have been applied. This level in the long run requires higher maintenance effort due to the duplication factors and the hand-holding.

Moving upwards requires:

- Enterprise Test Automation Strategy and Roadmap

- Capability Uplift and Training specific to usability, maintainability and scalability
- Pilot Automation Project

Level 3: Defined

Operating at defined level means test automation is planned and there is a level of reusability instilled within the test automation solution. This level of reusability includes such things like:

- Parameterised functions
- Object models that can be reused across scripts

Scripts are not just modular; they begin to make more sophisticated use of functions and common libraries, so that shared processes such as login, logoff are no longer embedded in multiple scripts; they become generic services or functions.

Test Automation is included as part of the testing phase and tests are designed and written with automation in mind. This level is a great starting point to maturing the capability within your project and/or organisation.

Level 4: Management & Measurement

This level is planned before testing starts. Test Automation focuses on ROI and a robust framework is developed that applies reusability of code and automation libraries. Test Automation covers many layers of the architecture model such as UI, Integration (Services and middleware) etc.

Level 5: Optimisation

The most mature level of all. Characteristics of this level are:

- Software development techniques are integrated into test automation processes – BDD, TDD
- Complete continuous integration and continuous delivery solutions are in place and operating as required
- Automation is essential to delivery
- Tests drive the documentation and development activities (TDD and BDD)
- Tests appropriately focus on the UI, Services and Code (Unit)
- Great test coverage – confidence in automation over manual
- Tests used in multiple environments and phases of delivery

Processes are reliable, reusable and repeatable at this level.

Appendix B: Example Business Case

Major time pressure for industry testing. Key factors into business change:

- Automation will help the project deliver to these time frames
- Risk and criticality were high, helps 'visualise' benefits

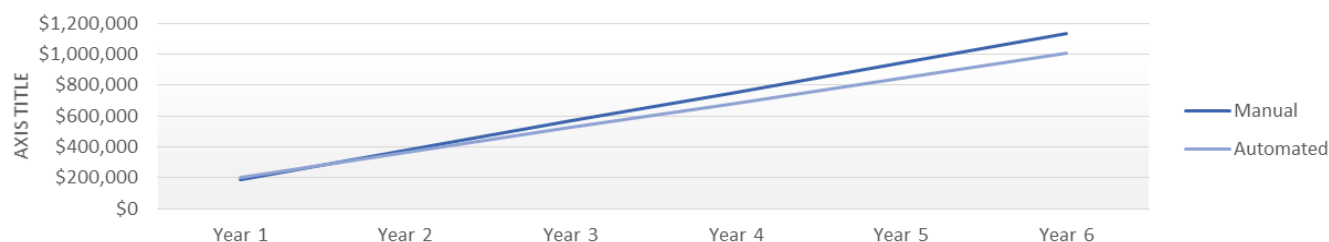
The Programme will undergo many releases of software that will impact the new software/solution

- Changes being applied to the UI and UX designs were still being agreed – this was being impacted by a third-party vendor
- Duration of the business change was set to take 12+ months
- ROI improves because the cost of developing the tools is offset by the number of times you can use or execute the test assets
- Business API's were developed and no changes were expected – these were ready

Long Term ROI


Automation ROI Inputs	#	Rate Per Day	Days	Notes
Planning:				
# of System Test Case	120	0.25	30	
# of end to End Test Cases	100	0.2	20	
Function #'s	20	0.25	5	
Object Repository			4	
% Scripts to be updated per run (10%)		10	11	
SubTotal			70	
Execution:				
# of Executions Per Release	6	100	6	
# of Releases Per Year	3		18	
# E2E (in addition to planning)	50	100	0.5	
# Legacy E2E	0	30	0	
Rates:				
Automation Resource Rate (\$K/day)		720		
Manual Resource Rate (\$K/day)		560		
Manual Calculations:				
Run Rate Per Release		8	112.5	
Run Rate Per Year			337.5	
ROI Results				
		Manual	Automated	
Cost Per Release		\$63,000	\$53,568	
Cost Per Year		\$189,000	\$160,704	
Initial Investment		\$0	\$42,480	
Cumulative Cost				
Year 1		\$189,000	\$203,184	
Year 2		\$378,000	\$363,888	
Year 3		\$567,000	\$524,592	
Year 4		\$756,000	\$685,296	
Year 5		\$945,000	\$846,000	
Year 6		\$1,134,000	\$1,006,704	

Break Even Analysis





NEED ADVICE? ASK OUR TEAM OF EXPERTS

 FUNCTIONAL TESTING	 TEST AUTOMATION	 PERFORMANCE TESTING	 AGILE TESTING	 TESTING TRAINING	 SERVICE VIRTUALISATION						
 MOBILE + APP TESTING	 DEVOPS SOLUTIONS	 SECURITY TESTING	 INTEGRATION TESTING	 COMPATIBILITY TESTING	 ACCEPTANCE TESTING	 WEBSITE TESTING	 OPERATIONAL TESTING	 APPLICATION MONITORING	 SAP TESTING	 GEO-LOCATION TESTING	 DIGITAL TESTING
 ACCESSIBILITY TESTING	 TESTING TOOLS	 ON-SITE TESTING	 TESTING CONSULTANCY	 MANAGED SERVICES	 COACHING + ADVICE	 TEST MANAGEMENT	 PROCESS IMPROVEMENT	 CLOUD SOLUTIONS	 TEST HEALTH CHECK	 BUG HUNT	 TEST STRATEGY

testing consultancy // training + certification // test tools + solutions

AUSTRALIA
1300 992 967
infoau@planittesting.com

NEW ZEALAND
0800 752 648
infonz@planittesting.com

UNITED KINGDOM
0203 457 5020
infouk@planittesting.com

www.planittesting.com