PwC Service Delivery Centre

Automation Test Guideline

[Document Revision History]

Ver. No.	Ver. Date	Prepared By	Reviewed By	Approved By	Description	PIO No.

[Template Revision History]

Ver. No.	Ver. Date	Prepared By	Reviewed By	Approved By	Description	PIO No.
1.0	2018-09-21	EPG Team-My Quality	Competency & PQA Owner	PQA Head & Competency Head	Initial Version of Integrated Template	0000037

Table of Contents

1. Scope of this document

The purpose of this document is to provide a guideline of the recommended approach and the best practise to automate the test activity when developing a software project.

It should be clear that this guideline does not discuss the business process automation nor does this guideline detail the RPA tools such as UiPath or Blueprism.

Moreover, this guideline is not a contract template. It contains a best-practiced document for generally agree-upon test automation.

2. Analyse and Confirm the Automation Test Requirement

No matter how much you want to discover and initiate test automation in your organization, you cannot do anything if your client is not convinced about the benefits test automation offers. It is a universal fact that test automation is expensive. The tools are expensive (HP QTP/UFT license cost around $8K per machine). There is a cost of a test automation architect or an engineer (which, by the way, is expensive too). After that, the benefits of test automation cannot be seen immediately. You have to wait 2-3 months before your scripts are prepared and tested and can run reliable for you to test the application.

You have to convince the client to bear the pain of these expenses and also you have to tell them to be patient before test automation starts giving them results.

So how can they be convinced? You have to tell them the cost-benefit analysis. For example, you can ask questions such as how much time we would take to test the BAT (Build Acceptance Testing) of our application. Then you can say, if it takes a day, then with test automation we can test it within only 2 hours. The cost is that you have to purchase the tool, train the resource and wait for the results for two months. After two months, we will be able to run a BAT within 2hours. That will save 6 hours of manual testing each time whenever a new build releases. If a build is released 4 times a month, you will be able to save 24 hours or 3 days of manual testing! In addition to the increase in test execution speed, there are many other important benefits to convince your customers.

- Process of automation reduces operative costs and increases both speed and reliability of task implementation, development and support.
- It enhances work continuity and satisfies the demand based on the market needs.
- It improves strategic analysis and streamlines the deployment of applications through automatic job execution using CI/CD process from Dev to QA, QA to Stage, Stage to Prod.
- It eliminates manual execution of command sequences caused by human error and increases efficiency and productivity of the organization.
- It enables visibility and control of all workflows and tasks and provides status reports of completed processes, the ones in progress and incoming tasks.
- It supports a local, virtual and hybrid environment in the Cloud and increases the possibilities of management and process control.
- The automation script can even execute on a schedule manner in any of the Cloud and Desktop environment.
- The automation script can even kick-off from individual test case like VSTS, TFS, QC and etc.

However, it doesn't mean that manual testers will not be doing anything. They will use this 6 hours of testing to focus on some new and important functionalities of the application while automation will take care of the regression issues. This setup will overall improve the quality of a product dozens of times.

If your client is not willing to pay for the quality of their products, then nobody can force them to do so. They will learn when they complain about the products. Quality affects everything. It affects your sales, your relationship with clients and your perception in the minds of consumers. So, intelligent clients have always invested in the quality of their products.

Five points to remember about convincing your client:

1. Tell them about the benefits of test automation in detail.
2. Tell them that test automation is expensive and it will cost you money initially but then the cost will be reduced once scripts are prepared and start executing.
3. Tell them that they have to wait for days/weeks before expecting any result from test automation. The time period highly depends on different projects and the number of test cases in the applications.

4. Tell them that test automation is not to replace manual testers but to aid manual testers as they will be able to test more at the same time.
5. Test automation does not mean more testing within less time. It means more testing at the same time (If manual testers get used to test the BAT in 8 hours, they will be able to test the BAT plus new functionality plus many other things within 8 hours in the presence of automation).

Remember, convincing your client is the first and most important step in introducing test automation in your project.

3. Formulate test automation strategy

3.1 Structure

Formulate one test strategy for all your automatic tests modelled closely on the Test Pyramid. Prioritize creating unit tests to cover most test scenarios over any other types of tests. Keep brittle UI tests that are tough to maintain at a minimum. Avoid repeating tests - collectively build test suites and avoid retesting a scenario if it is already covered in another test suite. Revisit and adapt the test structure periodically and refactor tests to keep your Test Pyramid in shape. Calculate overall test coverage including all kind of tests which can provide confidence during build & release.

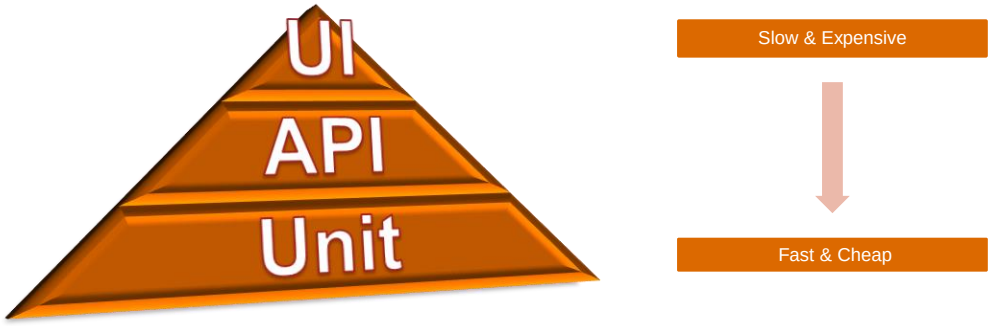

Figure 1. Test Pyramid

- **Unit tests**: They are fast running tests that are quick to debug. They should cover the bulk of your testing strategy.
  - Write unit tests for each and every test case - for models, controllers and views.
  - Typical speeds should be around 5 minutes - with a maximum of 10 minutes.
  - Cover maximum scenarios using “fast” test methodologies/types. For instance, write JavaScript unit tests to test client side validations, and avoid Selenium/Capybara based upon acceptance tests.
  - Is hitting database okay in unit tests? Yes, if they are pretty fast. However, by hitting the database in unit tests, we lose the ability to run unit tests parallel execution.
- **Integration tests**: They run slower than unit tests and they test component integration points.
  - These tests cover interactions and contracts between the system and external systems.
  - Typical speeds range from 5 - 15 minutes.
- **Acceptance tests**: These are end-to-end (E2E) test cases and consist of happy user journeys and critical failure scenarios which cannot be tested by unit tests.
  - Keep the numbers down - ask yourself: should I write an acceptance test - if I can write a unit test for this scenario?
  - These tests should be in production-like environment using artefact build for deployment.
  - They run between 10 - 30 minutes.
  - They help to verify that the E2E integration flow is working.
- **Performance tests**: These are long running tests and need to be run multiple times a day within the acceptable time frame of a critical production fix.

3.2 Speed

The definition of fast is debatable, and the team should collectively decide the acceptable time for the test suite to run and define it for all test types - unit, integration and acceptance. To help decide the timeframe, ask yourselves:

a. How frequently does your team check-in the code (remote push)? In the case of 2-3 times a day, test speeds of 10 minutes is acceptable. However if check-in happen 20 times a day, 10 minutes is not acceptable. Making the test suite run 1 minute faster can save 1000 minutes collectively across the team.
b. How much time is acceptable to roll-out a critical bug fix in production (including your Continuous Integration pipeline)? Lower is better, but don't kill yourself in speeding up your tests. Be pragmatic - up to 30 minutes is acceptable from check-in to deployment in most of the projects.
c. Can you run tests in parallel? One of the easiest ways to make the test suite run faster is to run them in parallel. In particular if you have unit tests that hit the database, you will lose the opportunity to run it in parallel.

3.3 Robustness

Tests should NOT be fragile. Tests should not fail without reasons. Re-running tests should not pass. Watch more closely on following kinds of automatic tests:

- Time dependent tests. These are the tests that run successfully when written but fail randomly based on the time of the day or start failing after some time. Here is some help on how to write such tests.
- Browser based acceptance tests. Due to the state, OS specific behaviour and random performance of the browsers, these tests keep failing most of the time. Using headless browser option is preferred, which provides stability and predicted behaviours. Avoid testing JavaScript behaviour and scenarios heavily using Selenium based acceptance tests. Instead of these, using Jasmine based pure JavaScript unit testing are more robust and fast.
- Test depending on external systems or services. If there is any kind of your tests depends on services and systems which is out of control from test, it should be mocked.

3.4 Debugging

When it fails, it should point to broken code. When a test fails, if it takes long to find the reason, then something is wrong with the test. Using test as your debugging tool as well, since automatic test is the quickest way to run specific scenario.

- Tests could be trying to test so many things and coarse grain. Define the unit under the test and use TestDouble to isolate unwanted dependencies.
- Bad test data doesn't help to identify root cause. While writing test, please avoid giving test data such as a,b,c or 1,2,3 etc. In such cases, it is very difficult to understand the failing use case.

4. Create the test automation pilot code and develop an execution plan

4.1 Create the test automation framework

The biggest task for the automation architect is to come up with an automation framework that should support automatic testing for the long run.

Basically, automation framework is a set of rules with careful planning to write the scripts in a manner which results in the least amount of maintenance. If anything changes in the application, the scripts need little or no updating to cope with that change. That is the charm of an automation framework.

There are five kinds of automation frameworks, namely linear, modular, data-driven, keyword-driven and hybrid.

4.2 Develop an execution plan

The execution plan includes selecting which environment the scripts will be executed. The environment includes OS, Browser and different hardware configurations.

For example, if the test case demands that it should check the website in 3 browsers, namely, Chrome, Firefox and IE, then the automation team will write the script in such a manner that it will be able to execute in each browser.
This should always be told before writing the scripts because it will be taken care in scripts if the automation team knows it beforehand. The execution plan should also state that who will execute the scripts. Normally the automation team executes the scripts on every build, but it varies from company to company. Some managers ask developers to execute these scripts on their build before release and some companies hire a dedicated resource just for the execution. Some companies even run scripts in unattended mode - which of course requires no additional resource.

5. Find automation tools and human resource

5.1. Find automation tools

There are various tools in the market, but you have to select the ones which are best for your application. However, the most important things to consider while selecting the right tools are:

1. The expense of the tools must be within **budget**. The automation tools are really expensive, so the company should have the budget to purchase the tools.
2. The tools must **support technologies** used in your application. If your application is using flash or Silverlight, the tool must support it. If your application is running on mobile, the tools must be able to execute scripts on mobile. You can purchase a single tool that supports all technologies used in your application or you can purchase separate tools for each technology. **For example**, you can use selenium for your web applications, Appium for your Android applications and MS Coded UI for desktop applications. Whatever the decision is, the cost should be within your budget.
3. You must have **skilled men** who can use the tools or learn the tools in short time. **For example**, you have hired an automation architect who has only experienced in QTP, and you are purchasing a license for MS Coded UI, then he or she might not be comfortable using it. Tools are like good cars, and you must have good drivers to drive them.
4. The tools must have a **good reporting mechanism** to show the results to stakeholders after each execution.

5.2 Top automation testing tool list

ID	Name	Type	UAT
1	Selenium	Open-Source	Web app
2	TestComplete	Commercial	Desktop, Mobile, Web app
3	Katalon Studio	Free	Mobile, Web, and API
4	HP QTP/UFT	Commercial	Web app, mobile native, mobile web, API, ERP
5	Robot Framework	Free	Web app, Mobile, API
6	IBM Rational Functional Tester	Commercial	Desktop, mobile and web app
7	Testim.io	Commercial	Web app (Machine Learning)
8	HP Quality Center (HP ALM)	Commercial	Quality management
9	Telerik Test Studio	Commercial	Desktop, mobile and web app
10	QMetry Automation Studio	Commercial	Web app, mobile native, mobile web, web services, and micro-services components
11	Ranorex	Commercial	Web, mobile, and desktop
12	Appium	Free	Mobile
13	SoapUI	Free/Commercial	API
14	Subject7	Commercial	Web app
15	Cucumber	Free	BDD TDD
16	Apache JMeter	Free	Performance test
17	Sikuli	Free	Web UI
18	Perfecto Mobile	Commercial	Cloud Mobile

Table 1: Top automation testing tool list

5.3 Find human resource

There are two types of automation experts: Automation architects and Automation engineers.

Automation architects are experienced in different kinds of tools and they usually know the strengths and weaknesses of each tool. They will help the client in selecting the right tool for automation by carefully analysing the application and technologies used in that application. They will help to build the framework, designing the naming conventions and create rules for scripting. They will also assist in selecting which test case to automate first.

If you are able to find the right person for the automation architect role, then half work is done in achieving successful automation in your organization.

Automation engineers, on the other hand, are the people who will convert manual test cases into automatic scripts. They will work under the supervisor of automation architects and will be responsible for creating and executing scripts.

Some companies hire automation engineers from outside and some companies do in-house hiring by training their existing manual testers. Whatever the case is, the resource must be good at programming. He/she has to know especially about object-oriented programming. A combination of on automation architect and two automation engineers is great for most of the products.

6. Write Scripts

It's the right time now to start writing scripts when the framework is designed, the execution plan is made and the resources are trained on the new tool.

Scripts should be written in an organized manner with proper naming convention. The source code should be maintained in a source control to avoid code loss. Version control and history should be maintained. Test automation is just like software development. All best programming practices should be taken care while writing the scripts.

Guideline of Translating Manual Test Cases into Automation Scripts

6.1 Precondition

The precondition is a particular state of the background to be set for a certain step to be executed. This is especially important in two scenarios:

- **To begin the test**: For example, we need the browser to be available and launched (The username and password availability will be dealt with in a little while). Now, how to write the same thing in the automation world? Consider QTP. You have an option to either launch the browser using programmatic statements or you can use the 'record and run setting' dialogue to set the properties. Setting these properties correctly is very crucial. In most of times, this is the reason why a particular piece of code will work in a machine and won't work in others.
- **To execute a certain step**: For step 2, we need step 1 to be done. To do so manually, we can just wait until the step execution is done and the page gets loaded fully. Use the sync or wait for statements in your automation script to achieve the desired state.

Note: When you are running the same code for multiple sets of data, you would want to make sure that you are returning the AUT to the state that it should be before the next iteration starts.

6.2 Test steps

We can put the manual test steps into 3 categories:

1. **Data entry** – Data entry steps are where you are entering some information as an input to your AUT.
2. **Change of AUT state steps** – these steps are the ones that will cause a change to happen to your AUT. It might include opening a new page, a certain field being visible, an edit box being editable etc.
3. **Combination** – as the name implies, this is the combination of the above types. Take the case of a checkbox, it will make a certain field active when turned on. In that case, you are entering the value "True" for the checkbox field and it also results in a state of your AUT.

The pre-requisite to create an automation script through any tool is spending some time analyzing the tool as well as AUT. Try to see how both of them interact with each other. For example, QTP has 3 recording ways and each one works in a different way. If you know how it identifies objects, you would know which one to use and use it better. If you have a web app where QTP can identify the objects easily, you can use the normal mode. If not, you might have to use the analogue or low-level methods.

Automation steps:

Data entry steps are not very different from the automation and manual methods. All you do is to enter the data. The way you reference the field is different. Since it will be the machine to perform the steps, we just have to make sure we refer to the fields in the AUT in a way that the tool understands. That means you have to use its logical name as used in the code.

For Change of AUT/Combination steps in a manual scenario, you perform the action (clicking, checking or entering) and verify the change at one go. But it is not possible in an automation scenario. So we have to make sure we add steps for action and validation/verification.

Comments for readability.

Debugging statements – these are especially important for creating and testing the test itself. Try to use message boxes frequently to output various values at various stages of the test execution. This will give you a visibility into the test that nothing else would.

Output statements – to write to results or any other external place like a notepad or excel sheet.

6.3 Verification and validation

Without verification and validation, the intent of testing is lost. Typically you will use a checkpoint (does not necessarily mean the inbuilt ones). So you have to use a lot of conditional statements and also loop statements to build the logic.

An important thing to consider is the attribute that are based on your verification and validation should not be ambiguous. For example, for successful login, look for the display of inbox page instead of the number of new emails for that is not a constant value.

So you have to pick something that is true every time when a set of operations happen.

6.4 Test Data

The following are some of the questions that you might consider answering for your test data requirements:

- Where to place it?
- To hardcode or not?
- Security concerns?
- Reusability concerns?

When you look back at the manual test script, you will notice that having the test data, the username and password available is one of the preconditions to even begin the test.

6.5 Results

For a manual test case, you can put the result of each step in the “actual result” column. An automation tool’s result file contains the result of each step when executed.

Automation tools these days have very robust reporting features. However, you might still need to tailor the test results. So, include the steps to write frequently to the result files so you will know exactly what is going on while doing the execution. If the tool you are using does not support writing to the result file that it generates, it is a good idea to have at least an excel sheet or notepad associated with each test to put in comments about the execution status as you go.

6.6 Post Operation

Once you are done with testing, it does not need be explicitly mentioned in your manual test case to close the browser or close the AUT etc. As a tester, you would do it diligently. In the case of automation test, you can include these steps in your script. Cleaning up – is the best practice to take. Disconnect all the connections you created. Close all the apps. Release the memory.

7. Reporting

The reporting is usually provided by the tool. But we can create custom reporting mechanisms such as auto-emailing the results to the clients.

We can create reports at the end of each execution in the form of charts and tables if clients need. Clients should always be informed about the test case coverage, that is which manual test cases are covered in automation and which are remained.

8. Maintenance of test scripts

If best programming practices are followed and framework is good, then maintenance will not be a problem.

Maintenance usually occurs when there is a change request in an application. The scripts should immediately be updated to cope with that change to ensure flawless execution.

For example, if you are writing some text in the textbox through the script and now this text box becomes a drop-down list, then you should immediately update the script.

Some other kinds of changes include a change requesting an application which scripts are running on the English version of the application to also support Chinese. Your framework should allow you to update your scripts with little effort to support execution in Chinese too! That is why automation architects are really important.

If the framework is not good and the best practices are not followed, then maintenance will become a nightmare. Most automation projects fail due to poor maintenance of scripts.

Reference

1. https://www.softwaretestinghelp.com/automation-testing-tutorial-3/
2. https://www.softwaretestinghelp.com/how-to-translate-manual-test-cases-into-automation-scripts/
3. https://www.softwaretestinghelp.com/top-20-automation-testing-tools/
4. https://blog.testproject.io/2017/04/16/test-automation-best-practices/
5. https://www.thoughtworks.com/insights/blog/guidelines-structuring-automated-tests