BENEFITS OF USING AUTOMATION FOR SOFTWARE TESTING

Author: Richard Puncheon

BUSINESS & TECHNOLOGY CONSULTING

Contents

Introduction

The option of Automation Testing is a common debate for any project. The need to reduce timescales and testing effort to drive down project costs being the obvious attraction. However, unless automation is approached in a structured and professional manor it is quite common that a project will struggle to produce the expected benefit.

By following some simple guidelines based on identification of test scripts that will provide more ROI (Return of Investment) than others, and embarking on an automation framework that is defined by agreed standards – automation has a better chance of succeeding and the confidence in its existence within a project will tend to be higher.

This document serves to outline points; that when followed will provide any automation work within a project a clearer structure and outline that can be used to garner more benefit over time.

Automation Framework

One of the most valuable aspects of proceeding with automation on a project should be the production of an Automation Framework. In order to ensure that your project's automation is set up to produce the maximum benefit, it is worth taking some time out and setting the framework in place. Factors that form part of the framework include;

- **Identifying Areas of Automation that Maximize Return of Investment (ROI)**
- **Coding Standards**
- **Budgetary Restrictions**
- **Automation Tool Proof Of Concepts (POCs)**
- **Scripting Directory Structure**

Identifying Areas of Automation that Maximize Return of Investment (ROI)

The process of identifying scripts to be automated should be clearly defined, the actors and stakeholders of these scripts should be on-board and consulted – and the process should remain repeatable within each project, release phase or consistent timeframe.

Coding Standards

Any area of the project that requires an amount of development work – even with automation – should first produce a Coding Standards document. This document will ensure that coding techniques are followed and are consistent going forward to ensure that the code can be produced and re-used by other members of the team – and can also be easily translated if scripts fail.

Budgetary Restrictions

From the outset it is worth identifying what the projects budget is for automation, to identify what software and hardware will be required to fulfill the requirement and ensure that this falls within the budget. Obviously the sole long term aim of automation is to cut costs of a project – therefore the budget need not be set in stone initially (projects tends to 'invest' in automation from the outset), however you may find that you need to identify what tools you need, how many licenses and what potential extra costs are necessary to initiate automation at the beginning e.g. you may find you need a separate Add-In for the tool before you can even start to write test scripts.

Automation Tool Proof Of Concepts (POCs)

Coupled with the Budgetary Restrictions as mentioned above, part of setting a Framework in place is the identification and proof of why you have chosen the tools that you use. A simple Proof of Concept (POC) document should be written at the beginning of a project highlighting the tools trialed (you can use trial versions of most automation tools before you purchase them), against the intended AUT (Application Under Test). This will help to identify the suitability of a tool, any limitations and what, if any Add-Ins are required to be purchased and added to the upfront initial cost.

Scripting Directory Structure

Seems a simple aspect of the Framework – however good practice of setting out the automation environment should ensure that all users of the tool can effectively re-use and document their work efficiently. This will include factors such as identification of a suitable location to place a License Server (can it be easily accessible, is it big enough etc), clearly defined directory structure to place automation scripts and input and output data sheets, naming conventions of scripts, data sheets etc. These guidelines may form part of the Coding Standards mentioned previously.

Benefits & ROI

Automation v Manual Testing

There is no disputing that a manual tester that knows and understands the application possesses more intuition when testing the front-end or GUI of an application. The tester knows where to look, understands areas that have a high failure rate or that have recently undergone development work. The concept of automation testing is not to replace manual testing in any way, but to complement manual testing. Automation should look to go the extra yard where a manual tester may not be able to consistently test over long periods of time.

Automation test scripts that have been developed with effective robustness, that look to report on areas of known fallibility within the AUT (Application Under Test) and can verify and validate every field with every possible variation, will ultimately prove to replace the laborious element of manual testing.

There is a large overhead if a project asks of it's testers to validate and verify every aspect, every field, dropdown, list of values, every permutation within every field etc. One simple automation script could feasibly be written that checks each field on a screen and every possible permutation on that screen, including error conditions. Couple it with the possibility of running that script repeatedly after every functional release, bug fix release, maintenance release or even just daily during non-working hours – and you start to see that automation can report confidence on a daily level to perform tasks that were otherwise not cost-effective or even humanly possible to perform through manual testing.

It is therefore important to expect automation to not only perform tasks that manual testers would otherwise perform, but to also go beyond what is possible through manual testing.

Identification of Test Scripts – Manual Test Cases

The identification of manual test cases that are deemed to be suitable for automation is one of the pivotal processes in automation testing. There are various factors that need to be taken into consideration before the test script is even approached. It is always worth managing a 'League Table of Return of Investment (ROI)' for all work proposed. The factors that should be considered that will form the basis of the ROI include the following;

- **Effort – Manual Testing**
- **Complexity – Scripting**
- **Proposed Iterative Runs Per Cycle of Script**

Each factor will be related to the others, and placed together in a League Table, should form results that will identify the areas of an application, than when automated – provide the most benefit.

Effort – Manual Testing

Identification of test scripts that have a high manual testing effort rating should be considered. These will be the scripts that will look to decrease the amount of manual testing per cycle per project. If you have a script that takes only a short time to run manually, and the running of the automation script takes slightly less time, you will not see much benefit in duration, and therefore cost by producing a script.

If however, you identify manual tests that are laborious to perform and time-consuming – scripts that tend to consume the majority of effort hours per cycle per release – then the production of an automated script will reap more benefits per cycle over time. Thus saving time and shortening test windows.

Proposed Iterative Runs Per Cycle of Script

Taking the above Manual Testing Effort into consideration, if you have tasks that have a tendency to be repeatable many times over per cycle, an automation script that covers this area will increase the value of the script. If a script is developed that only covers a test area that is run once per cycle, then it will be less valuable than a script that is consistently repeated over again per cycle.

Complexity – Scripting

The last factor that defines the higher ROI of a test script concerns the complexity of the automation script to be developed. You may have a manual testing task that is proposed to be ear-marked for automation that is repeatable, however in order to produce it accurately it will require a complex script that is time-consuming to develop.

This factor shouldn't affect whether the script should be undertaken, however placing a complexity rating against the script may find that a slightly smaller manual testing task that is easier to produce will over time produce more benefit and value to the project.

With these factors in mind, we can then start to put a ROI League Table in place. Below is a simple table that takes a proposed script, then adds the following values; Manual Testing (in hours); Development Complexity (1 is the highest rating of complexity); Intended Iterations (how many runs the script is to be run per cycle).

As you can see, each script has a value in each column – with the ROI (Return of Investment) rating determined by the following simple formula:

(Manual Test Effort x Development Complexity) x Intended Iterations = ROI

Simply, the ROI with the highest value should determine how much benefit each script would bring a project.

Fig. 1

Script Name	Manual Testing Effort (hours)	Development Complexity (1 – High)	Intended Iterations	ROI Rating
Test Script 1	1	2	10	**20**
Test Script 2	3	4	5	**60**
Test Script 3	1	4	10	**40**

Regression Testing

Regression Testing is an area of manual testing that is the most obvious and common of testing tasks that is ear-marked for automation. Based largely on the assumption that regression testing is repeated consistently over a set period of time, and spanning multiple releases – it is easy to assume where the cost savings and ROI is determined.

This then forms the basis of test script identification, albeit subconsciously and without process and proof that the scripts will over time, produce an effective ROI. It still risks the failing that an automation tester will spend more time developing a script than it will save time, once the script is run – even with multiple iterations over multiple releases.

For example, a regression test case may take only 1 hour to run per cycle, with 5 cycles per project. With the total manual effort of the task, calculated to be less than 1 *effort day* in duration. If you then look to automate that task and it takes 4 hours to script it initially (due to it's complexity) – then 30 minutes to execute for each cycle for 5 cycles, it will look to take longer to perform that test than it did manually. Based on the fact, that this script does not have an effective ROI rating. It has a low manual testing effort rating and a high development complexity rating, pushing it lower down the ROI League Table.

This isn't to say that this area of regression testing shouldn't be addressed and automated, however it should be taken into account and identified that it is less of a

priority than other areas. Over time, smaller and less beneficial areas of the regression test pack – tasks that are at the bottom of the ROI League Table - can be addressed.

Agile & Automation

Automation forms the bedrock of the Agile working methodology. The need to constantly test an application from development, through testing and into production relies on automation testing in different forms and guises to reduce the amount of testing time, and to identify bugs earlier in the lifecycle.

Testing within Agile is performed throughout the lifecycle of a project. Therefore automation scripts can be developed that offer more re-use across different platforms and environments. For example, a simple end-to-end script that is run in a System Test environment may only serve to highlight one or two defects if it is run in a Test environment, however it may also be run before and after the functionality reaches the Test environment, therefore identifying bugs earlier or identifying areas that have regressed in a UAT environment. Each time the script is run in each environment it adds a degree of trust in all Dev and Test environments throughout the cycle.

It shouldn't look to stop there. Scripts can also be used to provide a variety of other functions. Examples of scripts that can be used by other teams for example - include using a script that can be run by Production Support that checks the points of an application that interface with other systems. This test, when run on a schedule (sometimes during non-work hours on a scheduler) can be used as an effective health check of an application that identifies a system or interface that may be down – that without checking will be reported as a Production Incident.

Automation can also be used to perform a variety of other tasks on a project. Anything that requires lengthy manual and repeatable effort, whether in Dev or Test - could be automated to cut down the time of human effort.

Misconceptions

As is detailed previously, automation is an area of the project that shouldn't be limited to traditional areas of automation e.g. regression testing, repeatable tasks. It can replace a variety of tasks that are deemed manual across the entire lifecycle of a project. Therefore the tag that an automation tool should replace regression testing is short-sighted and will restrict the benefit of the process. Automation should be seen to assist in any capacity across the project, as long as the tasks it is asked to perform have been identified and scoped out to gauge its overall ROI, whether in the short or medium term.

With this in mind, automation should also not be merely ear-marked for existing legacy systems that are stable enough for scripting to commence. Automation can be carried out throughout the lifecycle of a project and also on new systems. Using the process of identifying scripts that are most beneficial, you can start to script on areas of the application that are stable and bolt scripts (actions) together as functionality is added to the AUT.