Cyreath.co.uk

Empirical Pragmatic Testing

An Approach to Project Sizing & Complexity

Applying an agile flavour to sizing and complexity enumeration

Discussion Document by

Mark Crowther, Principal Test Architect

Marios Michaelides, Programme Manager

Table of Contents

Introduction

In the first half of 2014 we (Mark Crowther and Marios Michaelides) were called on by an international investment bank, to help manage the delivery of test automation to a large suite of applications. The delivery required us to work with a larger group of Project Managers, a select number of automation vendors and address a back-log of several hundred applications. The primary objective was to save multiple €MM spent annually on manual testing.

We proceeded to engage with the automation vendors by following an operating procedure designed to regularise the process of delivery. With such a large Euro saving goal and so many applications to process, this was essential. While establishing and utilising the operating procedure we also worked on refining the process and practice followed, as much of the detail around this was not in place for this new programme.

One problem with the practices employed kept being called out by the application owners – the schedules for delivery of test automation provided by the automation vendors were way off. Typically, the level of effort planned for was significantly under that which was required. Our role in this regard was to find out why and to introduce a solution.

Feedback and Comments

Your thoughts, feedback, corrections, etc. are always welcome. Please find us on LinkedIn or via the website and be sure to name the paper or site content you are referring to.

Kind Regards from Mark Crowther & Marios Michaelides

Mark: https://www.linkedin.com/in/markcrowther

Marios: uk.linkedin.com/pub/marios-michaelides/15/bb5/276

The Estimation Problem

In projects with immature ways of working it's common to start by using the simplest approaches to a given problem. In our case it was how to estimate the effort and thereby the duration, required for delivering automation of a suite of manual test cases.

The simplest approach to this would be to ask; **how many test cases are there** to be automated? This will elicit a simple numerical answer that then gives a rough size to the problem. Let's say there are 100 test cases identified. How long will it take to automate these? Perhaps we can just pick a reasonable sounding figure of say "5 test cases automated per day".

The experienced tester or project manager will already have a number of 'yes but…' questions going around their minds.

Yes but…

- Not all test cases are the same length in terms of manual execution time
- Not all steps are as simple to execute as all others
- Not all test cases are independent of each other
- Not all inputs and outputs to the test case (pre and post conditions) are the same
- Not all testers can automate the same number of test cases per day

… and all the others you can think of.

The estimation problem then is not just one of counting test cases or test steps.

The issue is Complexity

As indicated above the issue of using a very simple estimation model (for anything), is that it doesn't factor in the various complexities that could be present. For example, if we divide our 100 test cases into 5 per person per day, we'll assume an effort of 20 days. Now what happens when we've automated the simple UI tests and moved onto the mid-tier and back-end tests? What happens is the schedule becomes erroneous. In this case it slips.

Even where we get ahead on the easier UI tests, it will create a false impression of progress as we'll now be ahead of the schedule. This again means our planning and estimation was erroneous and on both counts the schedule can't be trusted. What's more we'll not have factored in where the likely low and high complexity will be discovered next.

This was the effect of the estimation approach that we were called on to address. As test and project managers we must have a way to estimate, factoring in complexity and team skills.

Story Point Analysis

A key feature of delivery planning in agile projects is to determine the complexity of each bit of functionality to be delivered. These are broken down into Epics and Stories. On our projects we only had test cases, so we used them as stand-ins for stories. Some of the larger test cases may well have qualified as Epics, but as we wanted a way to introduce the more agile way of estimation, we took it one step at a time and just adopted the idea of a story.

In agile terms a story is usually in the format of “As a User – I wish to do something”. For us of course, we had step by step prescriptive test cases. Complexity will be defined in terms of story points and using these was the aim of our exercise. Separately the agile team will agree how many story points can reasonably be delivered in a Sprint of a given duration. This measure will then determine the size of the sprint in terms of test cases to be delivered.

Defining Complexity

The first challenge then, was to define complexity in numerical terms that could later be used to calculate story points. If we were to recognise it and enumerate it we needed to assign it a value.

Every test case to be automated was to have a complexity rating applied, which would be a measure of difficulty in automating it. From the complexity rating we could determine the story points applied to that test case.

Each test case would be classified using the following complexity ratings. Each rating has an equivalent set of story points.

Complexity	Story Points
High	13
Medium High	8
Medium	5
Medium Low	3
Low	2

The number sequence for the various complexities follows a Fibonacci Sequence. In this way we knew that as complexity increased, so would the expression of just how much more effort proportionally would be needed to deliver the automation.

Describing Complexity

With a complexity rating and number of story points agreed, the second challenge was in how we would describe and so recognise the complexity within a test case.

The vendor's automation team would be required to analyse each test case and on the basis of a number of factors to determine complexity. Of course in using this approach, we have to accept that this is not an exact science and there will be a degree of subjectivity in this process.

In thinking through the most likely and easily identifiable characteristics of the test cases, we settled on the following set of features as a way to identify complexity.

- Number of Steps
- Level of database interaction
- Systems Interacted with
- File Interfaces
- Quality of the existing script
- Pre-Conditions, such as data set up

This acted as a guide and checklist for the testers to use in assigning the story points. However, there was one more piece to the model that needed to be put in place.

Baselining Complexity

With the above agreed, we realised that a number alone was not a good point in space from which to go up or down a complexity scale. Just like 'number of defects found' doesn't tell you if the software is in a good or bad state. It might suggest so, but when it does you're working from some kind of baseline. That's what we needed, a baseline test case.

We selected a representative set of test cases that had been run for some time and asked the existing test team to rank them in terms of High, Medium and Low complexity. Once done we took the set to our in-coming test team and asked them to do the same. Those that matched a Medium complexity assessment were retained.

From this we had three 'exemplars' of what medium complexity looked like and we modified the set of features given above to include anything that was missing. Now, when those features were encountered in a test case the total number and the test case itself could be compared to our exemplars. Here's what a worked example looked like.

Feature	Description
Number of Steps	Up to 20 Steps
Repeating Steps	There are several repeating steps, which are broadly duplicates
Reuse of Code	This test case makes similar database calls across the different steps and so can be developed more quickly
System Interfaces	This has a single interface
File Input	Single Input file
How well written	The steps are well defined
Data pre-Conditions	The data set up is complex and varies across the steps

These criteria defined a medium complexity test case which we used as the benchmark to estimate the complexity of all test cases.

Calculating Effort and Duration

A problem we faced now was how to take the story points the approach so far allowed us to calculate and translate them into effort and duration, from which we could build our schedule. A quick search on story points will soon turn up the question of whether they equate to 'man hours' or effort as we refer to it as.

Strictly speaking, story points do not equate to hours of effort. Hours of effort and so the duration to deliver is a combination of story points and the velocity at which you can deliver them. On our project we had no history of velocity and again, this was agreed to be a step towards agile that was a bit too far for our nascent programme and recently engaged vendor teams.

We needed to take a pragmatic approach and so it was agreed, story points would equal effort in hours. Therefore, 1 story point = 1 hour of effort.

Calculate Work Effort

Now we're agreed that 1 story point = 1 hour of effort, all we need to do is map that against the complexity analysis we did for our test cases and check the number of story points we end up at.

We can then determine how many hours we have in a delivery period / iteration, multiply it by the number of team members to derive total hours of effort available, then create our schedule of delivery.

This is the objective of the model. To understand which test cases can be automated, given how many hours of effort we have available across the team, in the delivery period.

Conclusion

We've seen that while a simplistic model of estimation is initially attractive, it does not provide a way to meaningfully create a schedule of delivery. It lacks insight into the complexity of the tasks to be undertaken and so will result in schedules that are not realistic.

Switching to a hybrid form of story point analysis allows us to address complexity, but only so long as we define it first. When we've done that, we're able to combine the necessary analysis period by the testers into a planning exercise that delivers a useful and more accurate schedule of delivery.