Tips for manual testers working in an agile environment

Matt Archer

Tips for manual testers working in an agile environment

Matt Archer

This book is for sale at http://leanpub.com/tipsformanualtestersworkinginanagileenvironment

This version was published on 2014-07-23

Tweet This Book!

Please help Matt Archer by spreading the word about this book on Twitter!

The suggested hashtag for this book is #AgileManualTesters.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#AgileManualTesters

Contents

Introduction

Welcome to the review edition

If you are reading this page it means you have decided to invest some of your time and potentially your money in a book that is still a work-in-progress. Thank you for showing an early interest. After reading the pages that follow, if you would like to share your thoughts I can be contacted using the details below.

Email: matt@expresssoftware.co.uk

Twitter: @MattArcherUK

Hastag: #AgileManualTesters

As you read each tip, you will notice that there is a corresponding illustration. If you feel the urge to doodle on any of the graphics or create contrasting versions of your own then feel free to share these too. Feedback doesn't have to be limited to the written word.

The more observant amongst you may also notice that tip 49 does not have a title and I am truly unsure what it should be. Some time ago I arbitrarily decided that the book should contain 50 tips. If you have any thoughts about what the 50th and final tip should be, I would be more than happy to hear your suggestions.

Acknowledgements

Section T.B.C.

Notes:

Personal support, including friends and family.

Professional inspiration (currently sorted alphabetically) including Gojko Adzic, James Bach, Michael Bolton, Lisa Crispin, Paul Gerrard, Dorothy Graham, Janet Gregory, Julian Harty, Elisabeth Hendrickson, Cem Kaner, James Lyndsay, Alan Richardson.

Other people who I have embarrassingly forgotten in this draft!

Why a book of tips?

Whenever I join an agile team I ask myself the following question. How can I provide meaningful, quality-related feedback in a way that is compatible with the values and pace of agile software delivery, whilst maintaining independence, diligence and predictability?

You would be forgiven for thinking that after many years of asking myself this question I would have converged on a common answer. Nothing could be further from the truth. The reality is that every agile team is different and must continually be the subject of various experiments and trials to help each team evolve a blend of testing practices that will aid them in achieving their particular testing goals.

It is for this reason that you are reading a book of tips rather than step by step tutorials for specific testing practices. It is through these snippets of information that I hope to seed new ideas in your mind that will inspire you to alter the way you work by trying something new.

Maybe that new thing will work for you; maybe it won't. Either way, you will have expanded your experience of testing in an agile team through your own successes or woes. This, in my opinion, is a great way to study the craft of testing and continue to improve as a tester. I wish you the best of luck in all your agile testing endeavours and I hope you enjoy the tips. You can read them in any order you like.

Why have you focused on manual testing? What about automation?

Section T.B.C.

Note:

There already exists countless resources that do a good job of explaining automation and more specifically automation in an agile context. I did not want to repeat them here.

Conversely, there are few resources that look at testing in an agile environment from the perspective of a manual tester.

The book does include three tips on the topic of automation. Their purpose is to help manual testers spot risky automation so they can identify points in time when manual testing may need to form part of the mitigation for failed automation.

Whilst there is no plan to address the topic directly, by focusing solely on manual testing, I would like to think that this book can also help dispel some myths about manual testing in an agile environment, including manual testing doesn't have a place in agile teams and every tester need to learn how to write code to be of value to an agile team.

Why all the pictures and graphics?

Section T.B.C.

Notes:

I didn’t want to have page after page of text.

I believe visual anchors help people remember.

It’s a cliché, but sometimes a picture is worth a thousand words.

Tips

Tip 1: Appreciate that an agile tester never blindly follows a tip or practice

Tip T.B.C.

Notes:

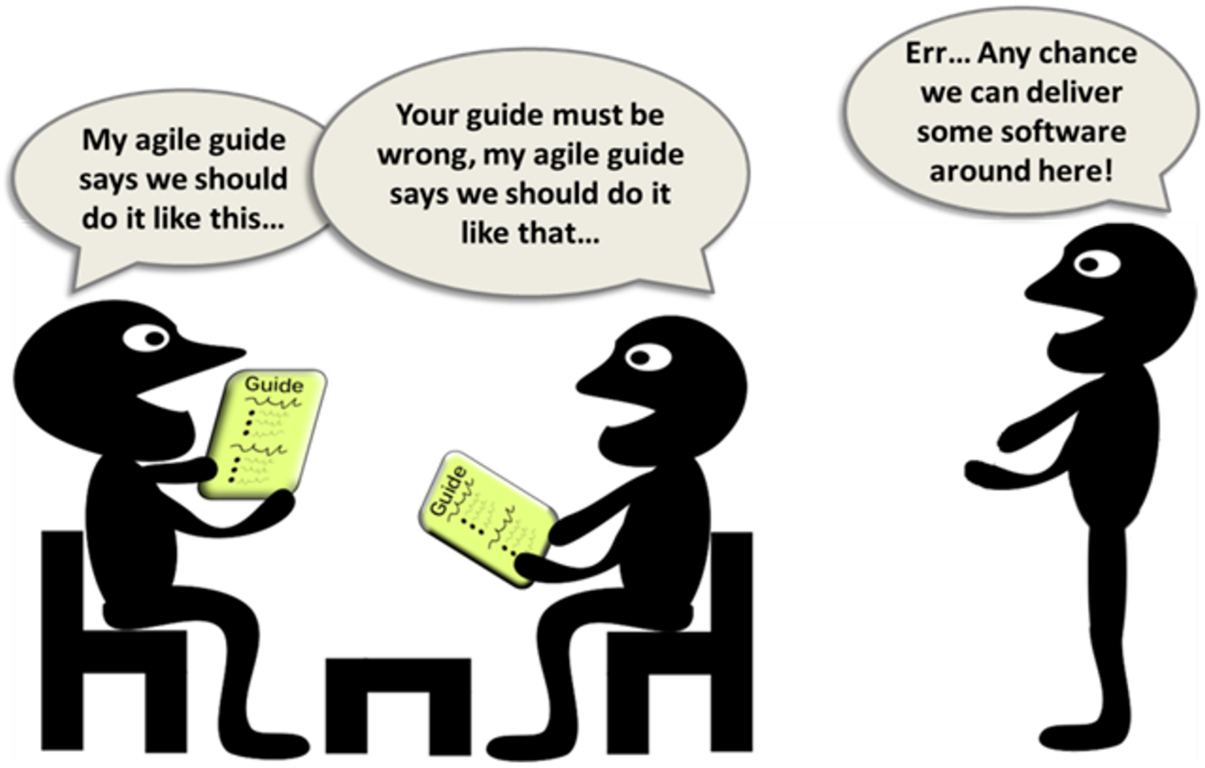
No piece of advice, including the tips in this book, should ever be followed blindly or without intelligently adapting that advice to something sensible that meets your own team's objectives.

Be careful that you are not introducing new ideas to the detriment of other practices in your team, both old and new. If you are replacing an existing practice, is the net result positive?

Sometimes you may feel that a particular tip or piece of advice doesn't apply to you or your team. That's fine, don't force it.

Trying to incorporate every tip in this book into your project is unlikely to end well. Take what is useful. Leave the rest.

Tip 2: Tailor your agile testing practices to meet your specific needs


My agile guide says we should do it like this...
Your guide must be wrong, my agile guide says we should do it like that...
Err... Any chance we can deliver some software around here!
Guide
Guide

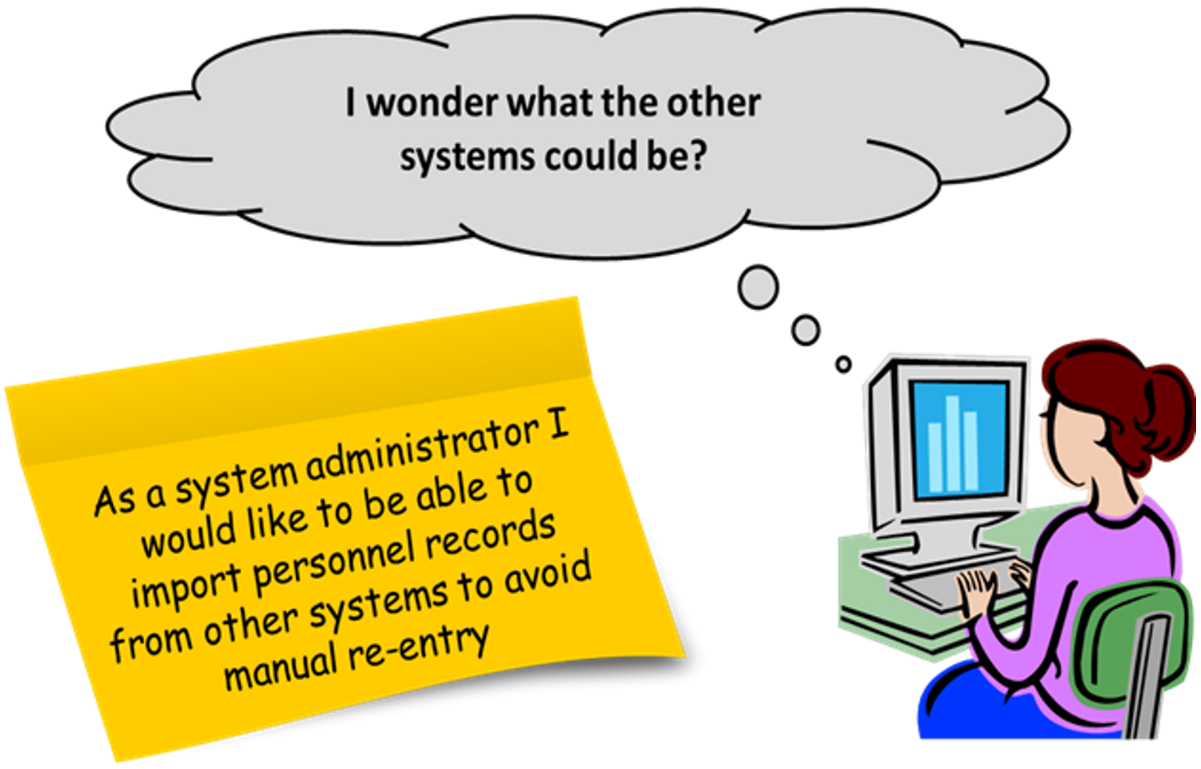
Some agile testing techniques appear so popular it can be difficult to keep them in perspective, alter them to meet our needs or choose to ignore them entirely. But this is what successful agile teams do daily.

It can be great fun debating the theoretical pros and cons of one approach against another, however, arguments both for and against a given approach are often invalided as soon as it is used in practice.

With this in mind, never be scared to try something new. Testers can waste days discussing whether a given approach is a good idea. This can be avoided by agreeing to perform a small trial. And by small, I mean as small as possible. Forget trying something for the whole of the next sprint, try it solely for the next story you build. At the very least it will give you a real example to discuss rather than the theory.

You may think that you don't want to compromise and instead stick to the agile ideal, but being adaptable is at the heart of agile. What follows is an extract from the extremeprogramming.org website. Many consider it to be the most important XP Rule. "Fix the process when it breaks. I don't say if because I already know you will need to make some changes for your specific project. Follow the XP Rules to start with, but do not hesitate to change what doesn't work."

Tip 4: When testing or preparing, don't allow yourself to be blocked


I wonder what the other systems could be?
As a system administrator I would like to be able to import personnel records from other systems to avoid manual re-entry

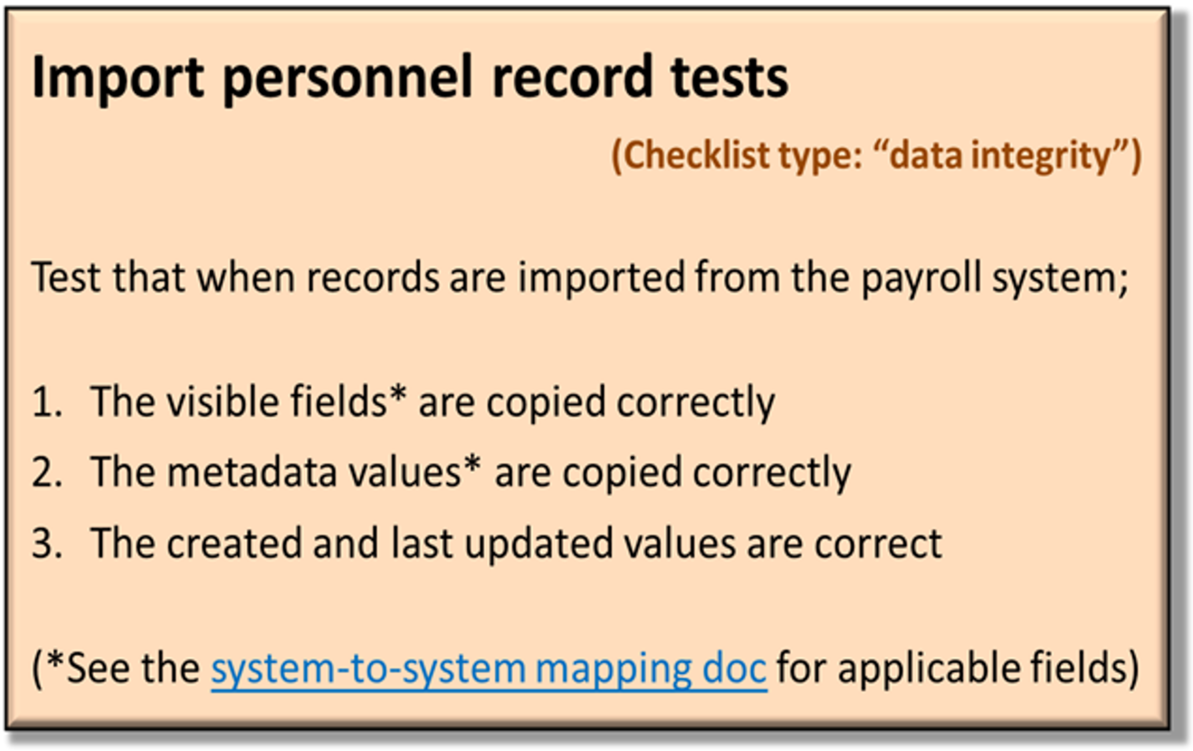
If your project aims to release their software ten months from now, losing a day because your work is blocked can often be dismissed as inconsequential. Contrast that to an agile team that aims to release every ten days. Spending a day idle becomes much more significant.

Traditionally, testers have placed strict entry criteria before their activities, but if we took this approach to agile testing we would spend the majority of our time underutilised, supposedly blocked. One way to improve your productivity is to discard the notion of needing a "complete specification". Instead, ask yourself what is the bare minimum I need to begin testing. The rest can come later.

Even when we begin to test early, it is easy to block or significantly delay ourselves by spending too much time pondering the gaps in our knowledge. When you encounter such a gap, resist the temptation to speculate for too long, especially in isolation.

If you believe the information is known within the team, now is a good time to remember the three 'C's that describe the journey of a user story; card, conversation, confirmation. If a conversation isn't available or doesn't leave you satisfied, leave yourself a note to revisit the gap at a later time and move on. Above all, don't allow yourself to become blocked, either by other people or yourself.

Tip 16: Share common information with other members of the team


Import personnel record tests

(Checklist type: "data integrity")

Test that when records are imported from the payroll system;

1. The visible fields* are copied correctly
2. The metadata values* are copied correctly
3. The created and last updated values are correct

(*See the system-to-system mapping doc for applicable fields)

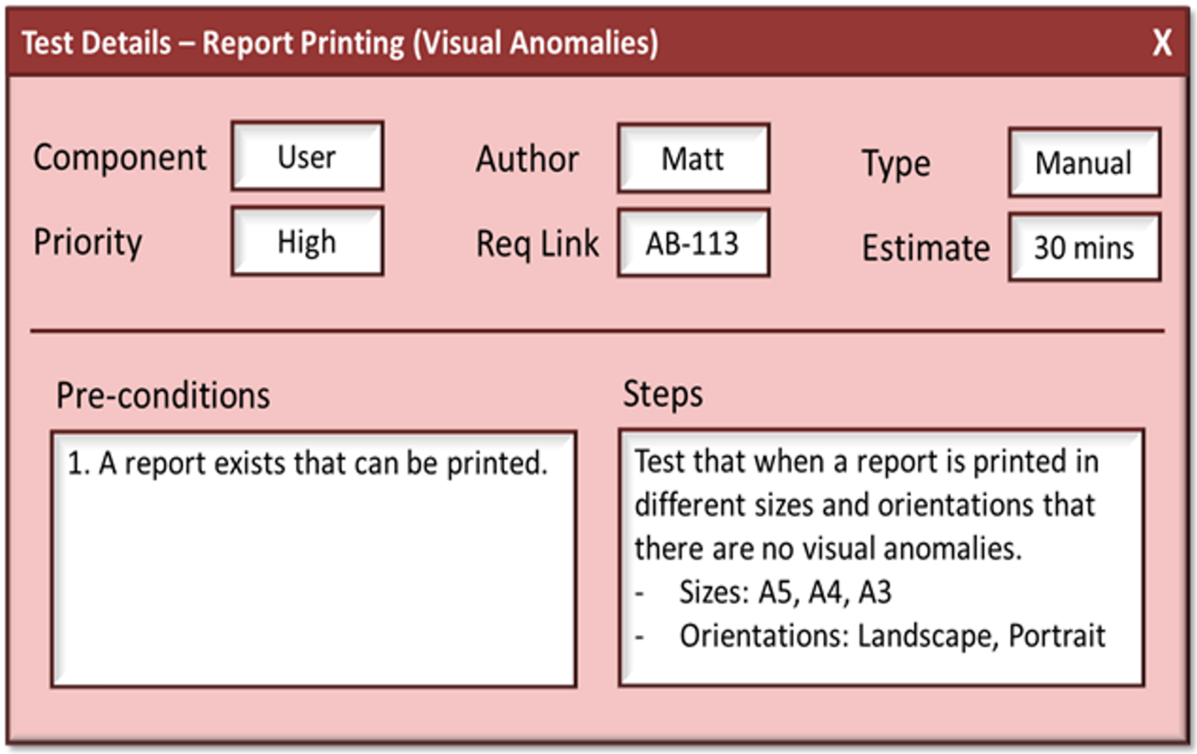
When testers work in isolation it is easy to forget that some of the information that we gather and store for future use is the same, or very similar, to the information that other team members document.

In agile development teams, testers typically work much closer with other team members which helps highlight this kind of repetition. That said, even when the overlap becomes obvious, the practice of "my documentation" is one of the hardest habits to break.

I attribute much of this resistance to the practice of using documents to represent key milestones and the rate of document production as an indicator of progress. Based on this mentality, reusing another team member's documentation can feel like you are not providing measurable value. In reality, this type of reuse will give you more time to focus on the things that make you valuable, like finding bugs.

This emphasis on documents is something that agile teams replace with a focus on working, business impacting software. Testers can help support this ethos by looking for opportunities to reuse existing documentation that have already been produced and either updating or amending it directly, or referencing existing documents from their tests or new (typically much shorter) test-centric documentation that only contains information that cannot be found elsewhere.

Tip 17: Use traditional testing tools in a way that makes you agile


Test Details – Report Printing (Visual Anomalies)

Component	User	Author	Matt	Type	Manual
Priority	High	Req Link	AB-113	Estimate	30 mins

Pre-conditions

1. A report exists that can be printed.

Steps

Test that when a report is printed in different sizes and orientations that there are no visual anomalies.
- Sizes: A5, A4, A3
- Orientations: Landscape, Portrait

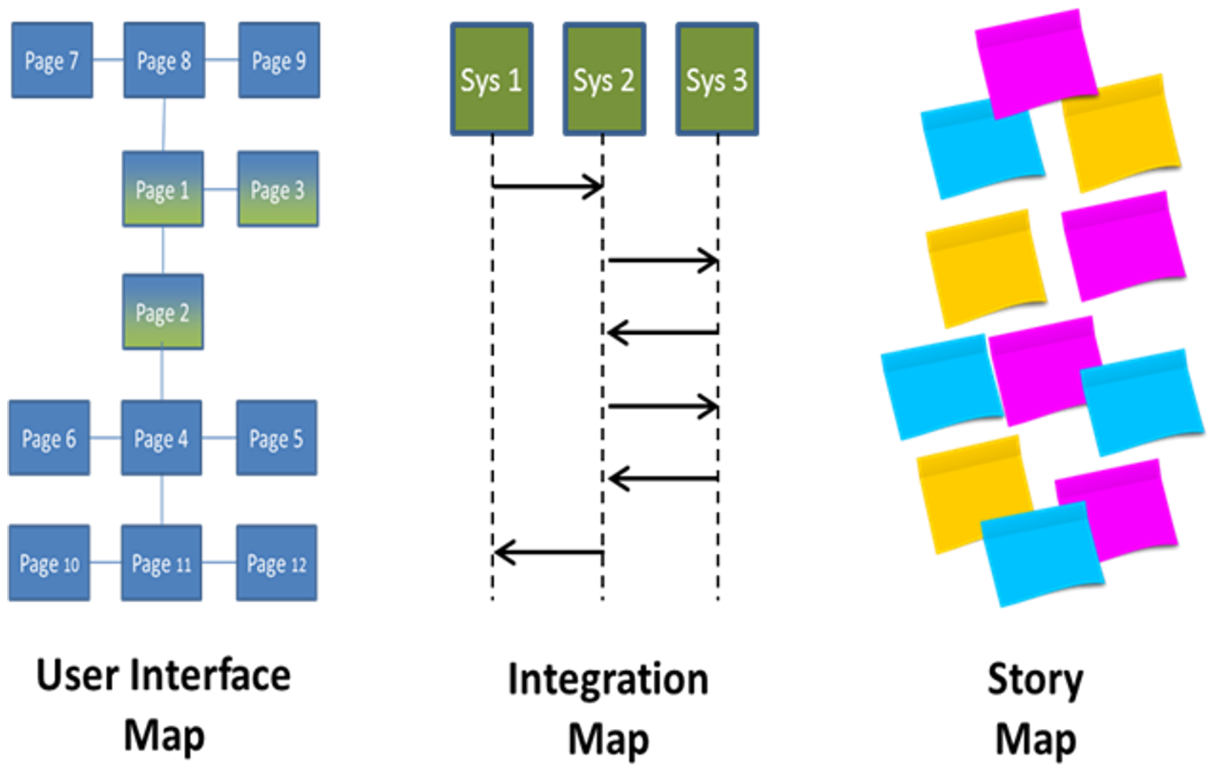
Some agile teams have the luxury of being able to choose the tools they use, but others are tied to particular tools or venders which may not necessarily be their first choice due to a lack of agile credentials.

If you find yourself in this situation, it is worth remembering that just because a vender had a particular usage pattern in mind when they created their tool, doesn't mean that you have to follow it. As an example, you could combine what the tool vender may consider different tests into a single record to reduce your test preparation and maintenance effort, whilst also placing related tests in context.

When you use a traditional tool in an agile way, it is not uncommon for people to worry about skewed metrics. In the example above, where we used to have 20 "test" records we may now only have five.

If you encounter this feedback, it can be useful to ask how the metric is being used. Are people assuming that an area with less "tests" can be tested in less time? I hope not. Or maybe that an area with more "tests" will be tested more thoroughly? An equally risky assumption. You can help your team by explaining how some of your tool's metrics have debatable value regardless of how it is used. It will also serve as a good opportunity to discuss what information your team would like and how you can accurately provide it.

Tip 21: Use self-generated maps to help organise your testing



To help organise our testing, we frequently divide software into parts. These individual parts can then become the focus for different activities, including test planning, designing, executing and reporting.

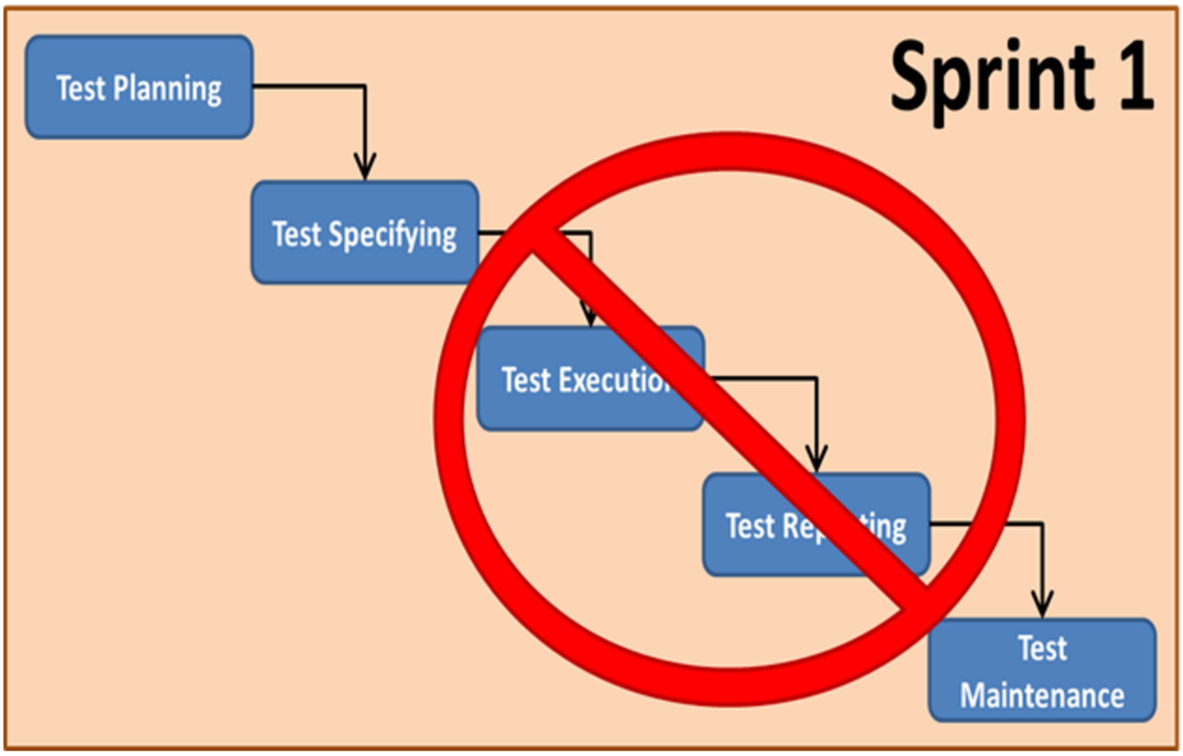
Traditional processes tend to dictate that testers should rely on artefacts created by other people to provide a map of the software and its different parts. A common example is to use a functional specification as a guide to partitioning the software's capabilities.

Whilst this type of information is useful, relying solely on models and specifications created by other people to guide our testing carries the risk of limiting our work to what others have committed to file. A risk that becomes even more prevalent when you consider that agile teams value working software over comprehensive documentation.

In the absence of a provided structure, there is the temptation to wander chaotically through the system looking for bugs. To do this is to overlook our ability to create our own abstractions of the system.

You can create your own maps using any tool or format. My only tip is not to be afraid of creating multiple maps or even temporary ones. Their goal is to help you consciously decide where to focus your testing next based upon everything you have learnt so far. Whichever maps help you with this task, are the maps for you.

Tip 24: Resist overlaying traditional testing processes onto a sprint



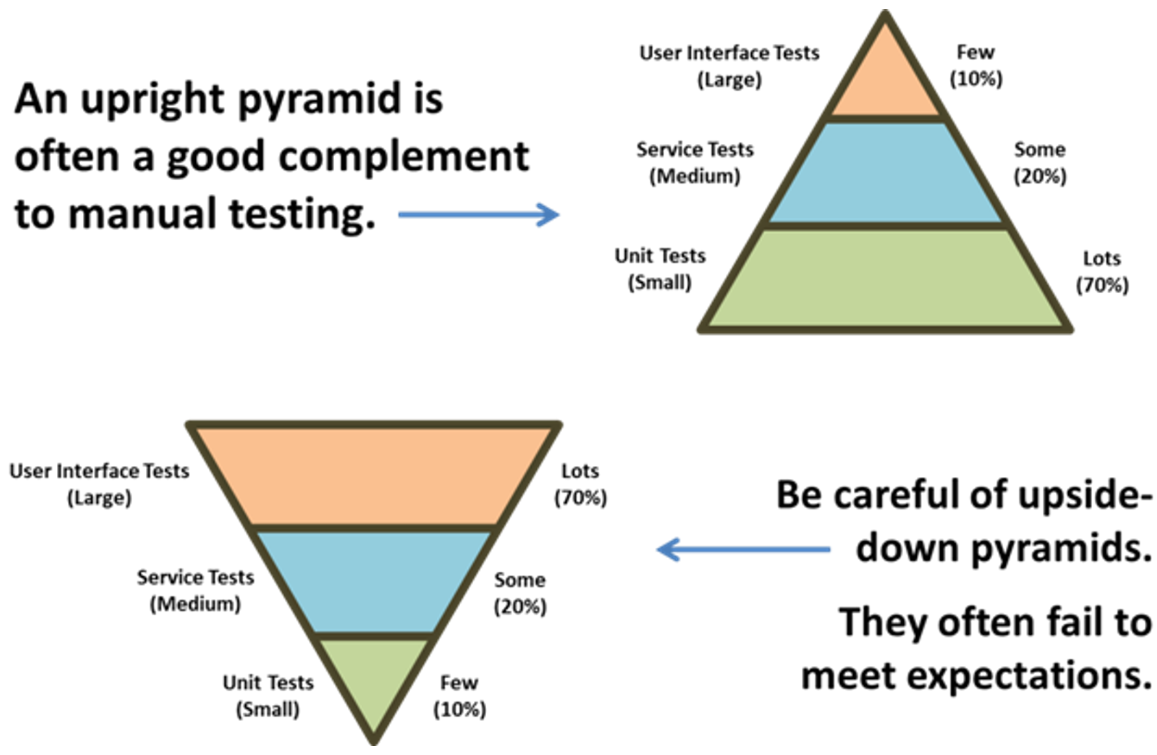
Agile projects rarely complete something in a single sitting, instead opting to progressively elaborate their understanding of the software and the software itself over time. It is useful to remember this aspect of agile when deciding how to perform our testing tasks.

It is not uncommon for testing to be organised around a planning-specifying-executing-reporting-maintaining pipeline, but this carries the risk of achieving failure in any project, agile included. In addition, testers who overlay this approach to testing onto a sprint carry the risk of feeling at odds with the rest of the team, to the point that even the most test-sympathetic teams can feel like a challenge.

One way to overcome this problem is to progressively elaborate our testing in the same way that the team evolves the software. This doesn't preclude an element of upfront test planning within a sprint or early test ideas, but be mindful that some aspects of the solution are yet to be decided and those aspects already decided may change.

This turns questioning into an finely-balanced art. Early questions are to be encouraged, but demanding a complete specification is unlikely to allow your team to work in the agile way it desires. Similarly, providing early feedback is rarely a bad thing, but raising a mountain of bugs against an early prototype as if it were a release candidate is unlikely to get you an invite to view future early work.

Tip 27: Learn how to spot risky automation: an upside-down pyramid


An upright pyramid is often a good complement to manual testing.
User Interface Tests (Large)
Few (10%)
Service Tests (Medium)
Some (20%)
Unit Tests (Small)
Lots (70%)
User Interface Tests (Large)
Lots (70%)
Service Tests (Medium)
Some (20%)
Unit Tests (Small)
Few (10%)
Be careful of upside-down pyramids.
They often fail to meet expectations.

The next three tips are about automation. Not because I think every tester should learn how to code, but because it is useful to be able to recognise automated tests that may fail to meet expectations.

The majority of agile teams create different types of automated test. Some focus on a few lines of code, others encompass chains of integrated systems. Some run in milliseconds, others minutes. Some interact via the user interface (UI), others probe underlying services.

When relying on automated tests, the most risky and unreliable tend to be those that cover the largest area, take the longest to run and interact with volatile aspects of our software, like the UI. We cannot always avoid these tests, but a team can aim to use them sparingly.

This is where the test automation pyramid can help identify risky strategies by suggesting a ratio between focused, quick, beneath-the-UI tests and broad, slow, against-the-UI tests. Predictably, the ratio is in favour of the focused, quick, beneath-the-UI tests.

If your team's automated tests go against this ratio, I recommend taking a closer look. If what you discover is of questionable value, discuss with your team how manual testing can temporally help, including any extra support and resources required to fill the gap.

Tip 32: However you document your manual tests, don't repeat yourself

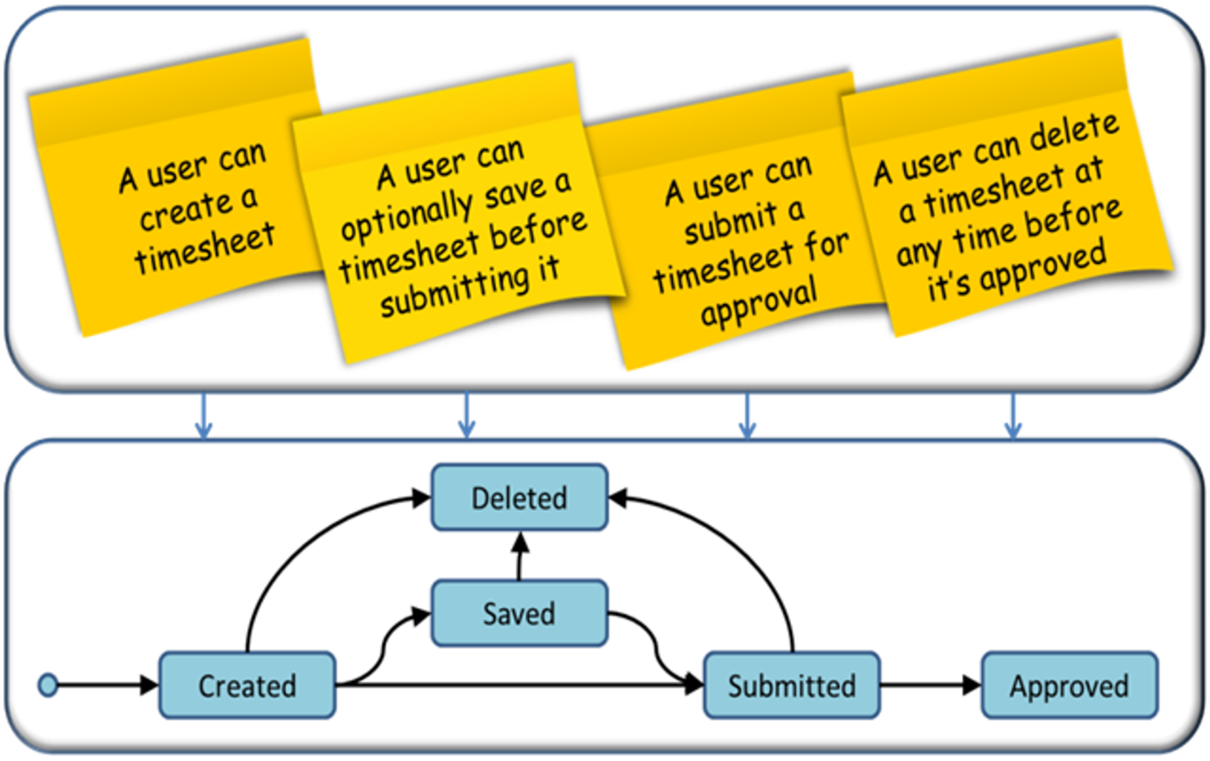
For years, programmers (at least the good ones) have known that when one piece of code looks very similar to another piece of code then this typically isn't a good sign. If something needs changing in the first piece of code, then the same change will almost certainly need to be made in the second piece of code too.

The more duplication that exists, the greater the chance that a mistake will be made during the update, the update will only be made in some places or changes will be rejected / ignored due to the daunting amount of effort that often comes with such duplication.

Unsurprisingly, when the description for one manual test looks very similar to the description for another then this typically isn't a good sign either. It generally isn't a good sign for any tester, but for a tester who is working in a agile team where 'responding to change' is valued over 'following a plan', duplication can quickly spell trouble.

So next time you find yourself reaching for the copy & paste icons, consider the duplication that you may be about to introduce into your manual tests. And more importantly, consider the future maintenance cost to you and the team.

Tip 41: Test multiple stories together to uncover different perspectives


A user can create a timesheet
A user can optionally save a timesheet before submitting it
A user can submit a timesheet for approval
A user can delete a timesheet at any time before it's approved
Deleted
Saved
Created
Submitted
Approved

Efficient agile teams are often compared to a well-managed production line, with new stories flowing quickly from a concept, through development and testing, before being rapidly released so that users can immediately benefit from their introduction.

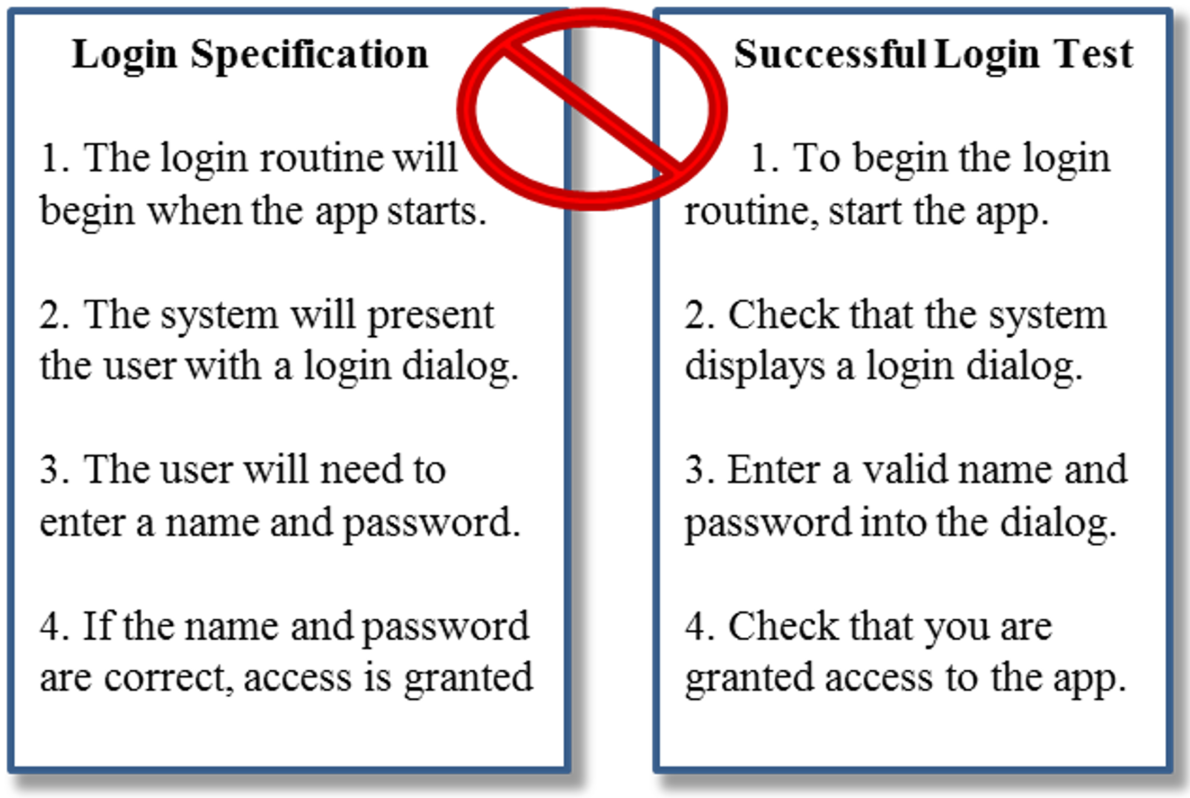
This type of efficiency gives the team and their clients a competitive advantage over slower agile teams and teams following traditional process models, but it also introduces some potential testing pitfalls.

When we test in a fast paced team, it is easy to become too focused on the individual units of work that progress through our production line (the features / stories), without considering the bigger picture.

To help mitigate this risk, I recommend thinking about each new story from a variety of different perspectives, including how it relates to stories already released. By combining multiple stories together you will typically unearth a more complicated assortment of inputs and interactions that can be the basis for wider-ranging tests.

That said, be careful not to lose focus entirely. It is important to supplement (not replace) our focused tests with tests that are based on a broader viewpoint. I say this because it is typically a variety of tests from different perspectives that help us minimise effects like inattentional blindness and find the greatest number of bugs.

Tip 42: When you describe your tests, don't just copy existing documents


Login Specification

1. The login routine will begin when the app starts.

2. The system will present the user with a login dialog.

3. The user will need to enter a name and password.

4. If the name and password are correct, access is granted

Successful Login Test

1. To begin the login routine, start the app.

2. Check that the system displays a login dialog.

3. Enter a valid name and password into the dialog.

4. Check that you are granted access to the app.

There is a an approach to describing tests that can be thought of as transformational. You won't find it described explicitly in any book but it is alive and well in many projects. The approach is simple. Find something that describes how the software is expected to work and transform it into the description for one or more tests.

The approach often begins with a element of copying and pasting from other documents, typically followed by altering a few words and on occasions the addition of either inferred or specific test data.

Unfortunately, following this approach will saddle your project with unnecessary debt. This is because it now has at least two places that describe how the software is expected to work, in a very similar way. Any changes to that expected behaviour must now be maintained in multiple places. Using this approach will also limit your testing to other people's thoughts and ideas rather than your own imagination.

To keep your debt to a minimum, avoid repeating information produced by other people. If your project already has a good explanation of how the software should work, reference this item to provide context for your test and instead of writing lengthy tests, opts for brief test ideas that make sense when presented alongside existing explanations of how the software should work.