Tales from the trenches: Lean test case design

By Darren McMillan
This article was originally posted at Darren's *Better Testing* blog on October 28, 2010.
Original URL: http://www.bettertesting.co.uk/content/?p=253

A short recap

In my recent post about proactive testing I promised to share my method for creating lean test cases. So as I like to keep my promises here it is; I hope you'll find it useful.

I'll start out with how this all came about, feel free to skip to the "lean in all aspects" section if you don't like listening to how people overcome their problems.

Some background

I used to dislike writing test cases a lot! I found it painstaking how much work was involved just to document what was in my head. I used to think this could be time better spent doing other things because I did waste a lot of time writing these - too much time - time that could have been spent testing!

So, like everything I thought could be done better, I started trying to think of ideas on how this could be improved. Obviously it starts out with a question - "What is the actual problem here?" The problem was my dislike for writing test cases & my desire to test instead! So, looking back now, it was obvious that I felt burdened by writing test cases on functionality which I already had available to test. I probably felt that for every test case I'd written I could probably have found some defects on that testable functionality.

Dilemma solved!

At the start of each new release I'd always have a bit of spare time which I could use to get other projects done. This time could have been spent writing my test cases, right? If I had some requirements I could start writing down test conditions. So that was it, I'd solved my dilemma of having to write test cases while all I wanted to do was test. I would just write them before the functionality was written.

Requirements review

So the release went out & the requirements for the next release came in. This was my chance to see if this made me feel any better. So out came the requirements & on went the thinking cap. I began jotting down my test ideas & looking at what types of testing would be required for this project. I also began noticing shortcomings in the requirements & things I knew just wouldn't work. I also began to generate ideas! Why do we do this? Or would it not be more user-friendly to do it this way? On and on it

went. I was having loads of fun.

I looked over what I'd noted after reviewing these requirements & noticed that half of it mapped out as an overview of how I'd test this functionality & the other half was good feedback for the BA's & managers to consider. I'd found that, on one hand, I could now easily write up my test plans & work out how much time I needed to write up test cases. On the other hand, I'd come away with a lot more useful feedback than I'd ever had before - some which might just make this project that little bit more successful if all went to plan. I was very pleased & most of all I was having great fun.

Introducing test cases early

Next stop was the test cases I'd hated writing before. This time around though, it proved to be very enjoyable. A few days later I'd written up all my test cases & while I knew they'd probably have to change a bit, I was quite happy in that not only was it more enjoyable, as they were being written I'd been spotting lots more gaps & potential issues with the requirements. The BA's really appreciated the feedback & I was happy that I was more involved with them, too.

So the release came and went. I'd begun my first steps into proactive testing without realising that I was being proactive. I'd noticed a big downturn in defects but I'd also spent a lot of time chasing developers to run these test cases I'd written. This seemed like a better way of working, making good use of a quiet period & contributing heavily to the team.

Lean in all aspects

So you're probably wondering when the lean part comes in right? We'll begin touching on it now, I promise

But how many test cases?

Previously everyone on our team appeared to have their own ways of creating test cases with no set standard. Some people opted for a test case for every one to three test conditions, resulting in hundreds of test cases. Others would bunch more conditions into test cases but for each step in the process that changed they'd always have to write a new test case. I didn't think this was very efficient, or fair at the same time for those people who'd have to re-run them if they ever became part of the regression test suite. I remember once two testers responsible for an important new feature got asked to write some test cases for it & came back with over eight hundred! Crazy & looking into these each had one or two conditions at most for each test case. How many test cases you've written means nothing, James Christie wrote an excellent article on this "But How Many Test Cases". Could you ever imagine asking someone to run those eight hundred test cases with a straight face? Crazy talk!

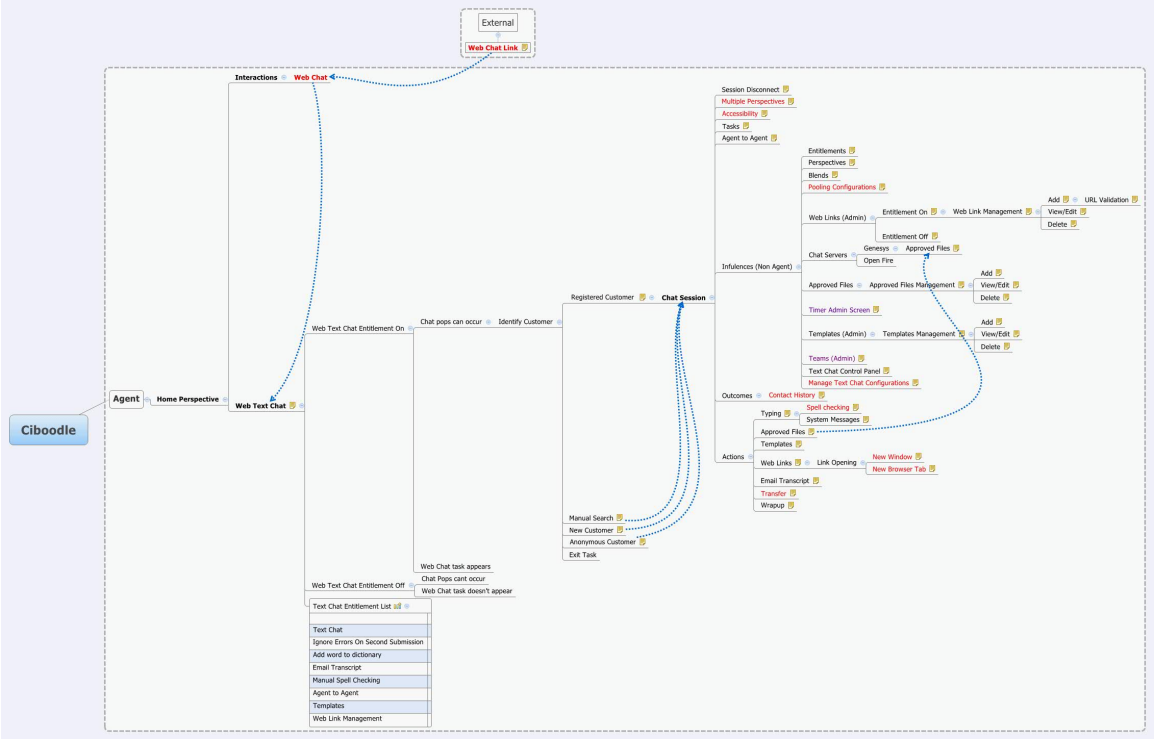
So that was the first challenge getting the rest of the team on board with how I'd thought these should be written. I'd arranged a meeting & asked that everyone to submit how they thought a test case should look. We discussed the pros & cons of each template & grouped our favourite couple, then discussed these some more. Thankfully, they all seemed bought into my suggested format.

Mind mapping

So what was the format? Well, it's evolved slightly since then so we'll just discuss what it looks like at present. Firstly, I think it's important to discuss how they are designed.

As I've said before, I generally I try to take a proactive approach by generating these up front. If you've paid any attention to requirements at the start of a project you'll know they can vary greatly from someone's vision to very detailed documentation. So you'll know the extent of your test cases will be limited by the available requirements, something which we've been lucky with at my company allowing us to provide very good test cases up front for developers to run prior to committing code.

Once I've given feedback on the requirements & they've become more stable, I'd crack open my favourite mind map tool, in this case Xmind. Mind maps are excellent. Previously, I'd been writing all my test cases in our test management tool & not seeing as much interlink between the different areas of the feature as my thoughts were constrained to my current test case. I'd also have to jump in and out of test cases whenever I'd thought of new conditions for a test case I wasn't currently creating or editing. With a mind map, you see all the links in the feature as you build it so you're fully aware of its integration points & influences.



With Xmind, you can add extensive notes to each node on the mind map. This is what I used to write up all my test conditions for that part of the feature, to later be dumped into the test management tool. If I think of a new test condition for some other part of the feature, I simply select and edit the notes of that node on the mind map. What took a little time before now takes seconds. Likewise, as everything is visibly in front of me, my mind begins to naturally consider more aspects of the feature. This, in turn, generates better test conditions that I previously might not have thought of when using a test management tool to write these.

Monkey Madness!

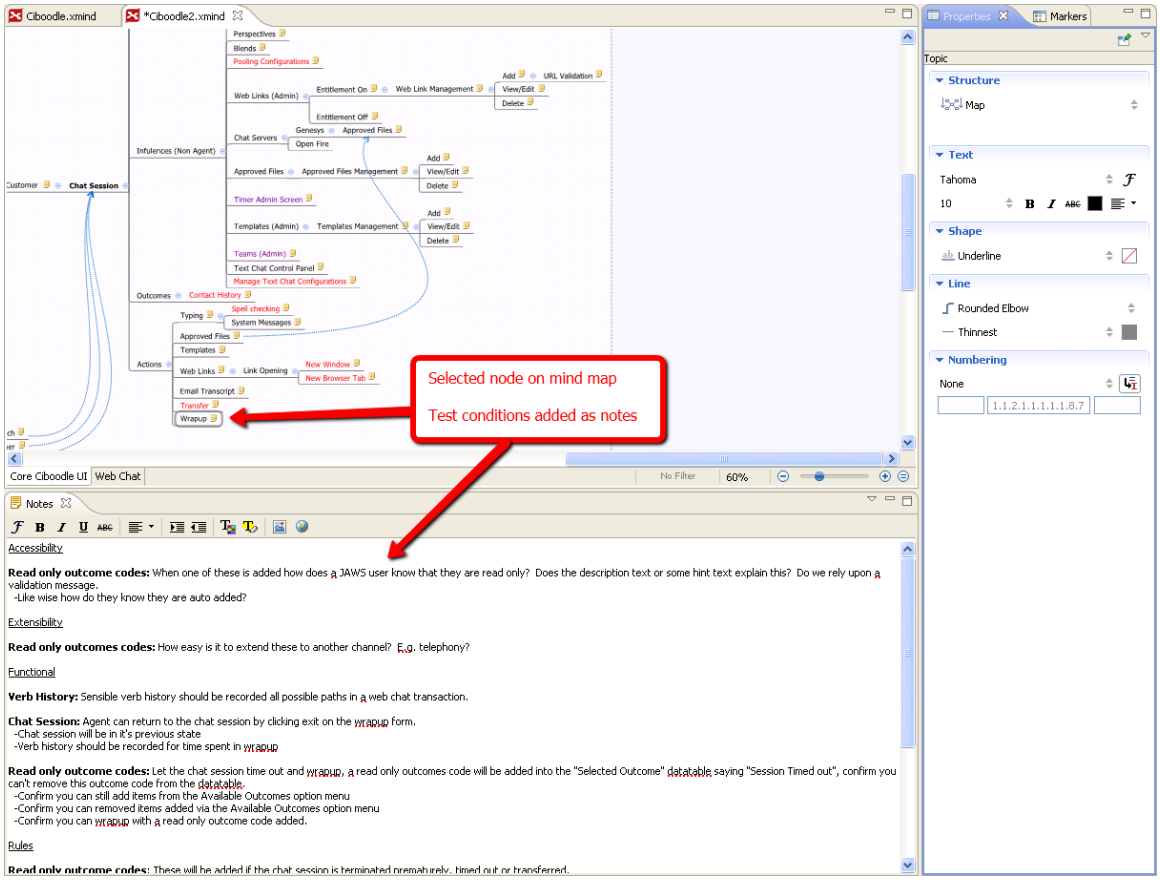
For me, my map shows my path through the application/feature, or steps as you'd call them in a test case. When it comes time to put my conditions into a test management tool, the map becomes the test case folder hierarchy, making it easier to navigate test cases when running them in the future. If my tool needed me to add steps for some reason (mine doesn't & I don't) I'd simply copy the folder structure onto my steps in a brief format e.g. Make Order > Add Billing Details > Confirm. I prefer to keep the fluff to a minimum so I don't use steps. People are intelligent & in my app, they can see that the folder structure mimics the steps they would have to take.

As you'll imagine with the nodes being paths or areas through the application or feature we can cover a broader feature area than before. We're not training monkeys by telling them "for this condition you'll need to select X, then after a second Y will appear" but to do that you need another step oh crap lets write another test case for this! No, we don't need to hold people hands. We can let people think for themselves. As long as the test

conditions make sense people will understand that they might need to deviate from the beaten path to test something.

Adding in types of testing

For the conditions I generate these on nodes on my mind map. I split them up into the types of testing I would do. When I say a testing type it would be something like Usability, Extensibility, Security and so on. I generally dump a generic template of all testing types into each node at the start as a note so I can at least attempt to consider if any conditions need to exist for that testing type. From splitting these test conditions into different types of testing, you'll find that you think more, as such generating much better test cases for that functional area, since your paying much more attention to what you'll need to test. I also add in a rules section for everything that the requirements expect, I guess you could call these our acceptance conditions.



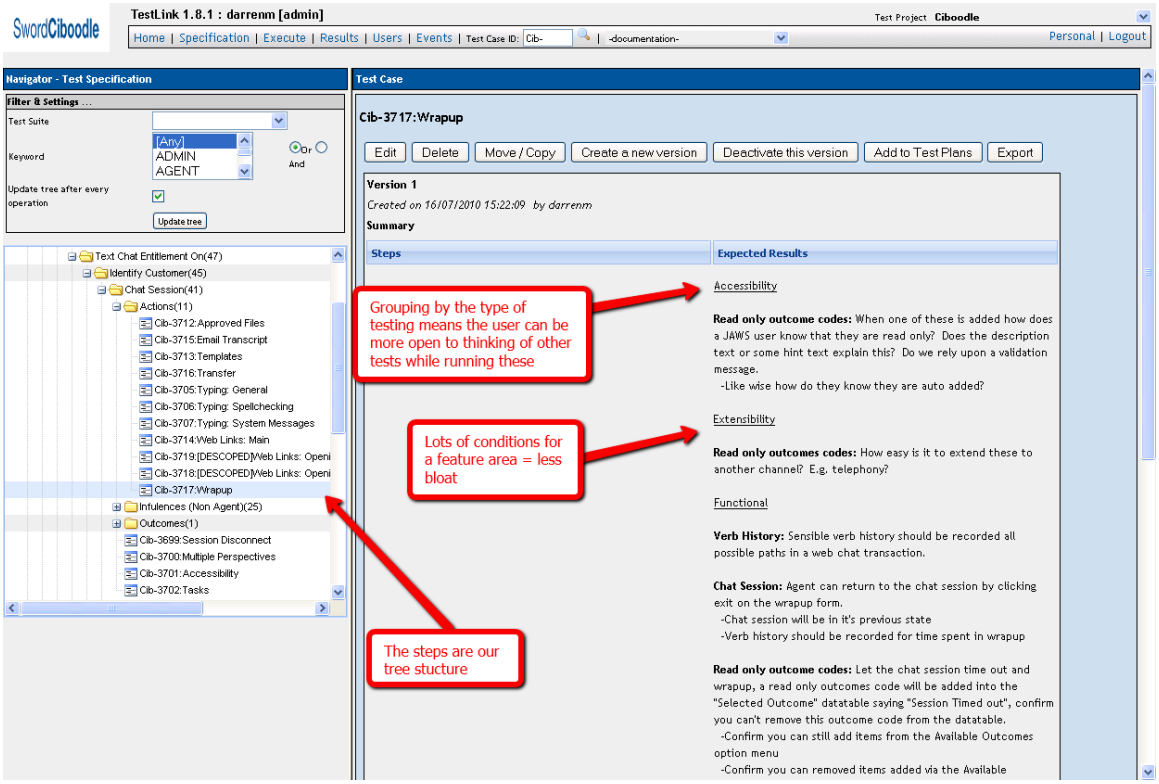
Increased requirements review

My final section at the end of my notes is for questions I might have about the requirements. The good thing about Xmind is that highlight sections with unanswered question in a different colour. Then, I can easily see that I need to resolve a few questions for that area at some point. Even if you've done a review and feedback of the

requirements, you'll find lots of other things you hadn't considered. At some point, you'll need to discuss those with a stakeholder. These might be gaps, risks or suggested improvements. Mind maps make it very handy to collect everything you've done in one place.

Generating the test cases

If you are thinking these are all very condition-based, you are correct! That's all I want. I'll take these once I'm happy with them from the mind map & put them directly into my test management tool as-is. Like I said, my steps will be my folder hierarchy or - depending on your tool - a very brief copy of the mind map's path to the node to replicate the steps the user would make. If I need some setup tasks done before running these test cases I'll put a link to them at the start of my conditions.



Requirements change. They always do & often, they are not strictly followed. That's why I tend to leave all of my test conditions in my mind map right up until the last minute. It's easier to change & add conditions in the map than in the test management tool. With the addition of a better testing mindset, using the mind map will make you more aware of the impact of a requirements change. As such, you'll be able to write better conditions around this.

I'd considered doing a demonstration of all this using a simple application but I think I've covered almost everything & hopefully made sense. If anyone would like me to do a

demonstration from requirements, to maps, through to the end test cases I'd be happy to, just leave a comment or drop me an email & I'll do this as a later follow up post.

Summary

So I hope from the description & the screenshots I've provided you'll be able to see the benefit of keeping your test cases as lean as possible. For a quick recap here's the why:

- Mind mapping
  - Increases creativity
  - Reduces test case creation time
  - Increases visibility of the bigger picture
  - Very flexible to changing requirements
  - Can highlight areas of concern (or be marked for a follow up to any questions).
- Grouping conditions into types of testing
  - Generate much better test conditions
  - Provides more coverage
  - Using templates of testing types makes you at least consider that type of testing, when writing conditions.
  - When re-run these often result in new conditions being added & defects found due to the increased awareness
- Lean test cases
  - Easy to dump from the map into a test management tool
  - If available the folder hierarchy can become your steps
  - Blend in easily with exploratory testing. Prevents a script monkey mentality.
  - Much lower cost to generate and maintain, whilst yielding better results.

That's it for this post, I hope you enjoyed it as much as I did writing it, thanks for reading.