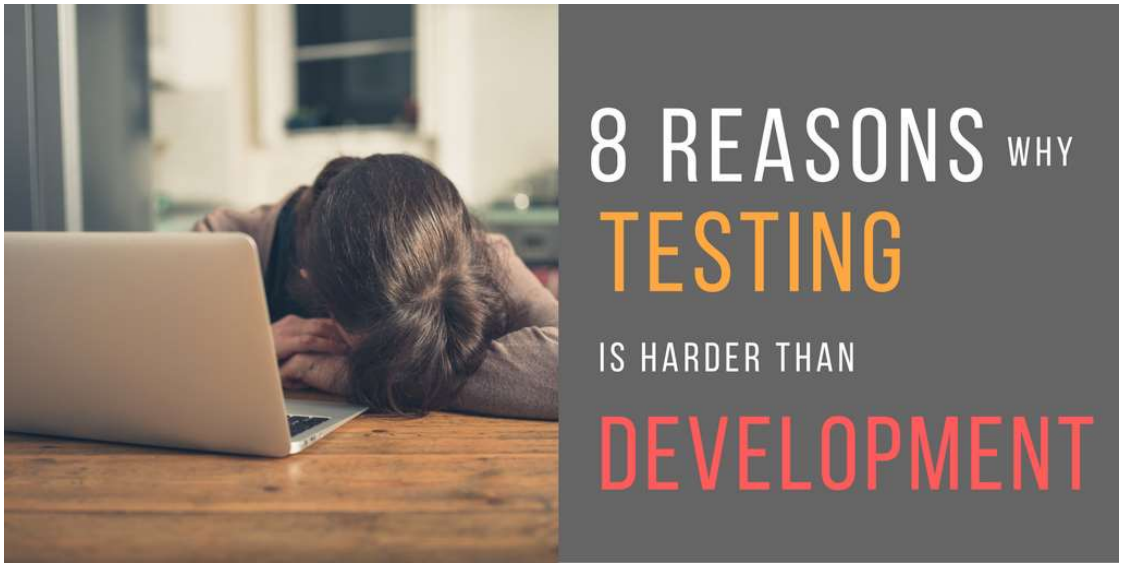
8 Reasons why Software Testing is Harder than Development

James Willett

27th October 2016



This is likely to be a controversial topic, and it's honestly a bit tongue in cheek, but the thought has occurred to me more than once recently. I figured that it would make for an interesting blog post and some interesting discussion. I think that, in the modern Software Development world, **Testing is harder than Development**.

If you had said this to me 10 years ago, I would have thought that you were crazy. Perhaps it wasn't true 10 years ago (it likely wasn't). Perhaps you think I am crazy to think this today, but in this post I am going to outline **8 reasons** why I'm not.

Let me make a confession. When I left University, I took a job in Testing because *I thought that it would be easier than Development*. I had struggled with writing much worthy programming code during the 4 year course. Even for the generic and mostly straightforward lecture assignments. I didn't feel that I had produced anything of decent quality. The thought of having to write ACTUAL code, to solve real problems in the real world terrified me.

But I knew that I was going to work in IT. At the time it was the only thing that I was passionate about. So I decided to *take the easy path of a career in Testing*. Let some other boffin come up with the complex solution I thought. I'll just run a few manual tests and tell him (or her) if it works or not. What could be simpler!

Well, 15 or so years later, it hasn't exactly worked out like that for me. *Thank heavens for that as well*. The job I described in the last paragraph doesn't sound like the most fulfilling. It's not something I could persist with day after day.

So why on earth do I think testing is harder than development? How can that even be possible? Let me clarify that for this post I am talking about the role of the modern tester. The tester who is embedded deep in the Agile team, flanked by developers. The SDET – Software Development Engineer in Test (or whatever variation your company has chosen to adopt). Not the old fashioned manual tester that I described above.

Let me make an important distinction here. *Crappy testing is easier than crappy development*. There isn't much doubt about that. In this post, I am saying that **excellent testing is far harder than excellent development**. This was highlighted by Neel Kumar in this great post on Quora . I obviously know that excellent development work is hard too. But I feel that sometimes *there is a perception within the industry that testing is significantly easier that development*. I don't think that is true, hence the reason for this post.

Without further ado, here are the 8 reasons why testing is harder than development:

Reason 1: You need a broader set of skills

So of course you have the core set of traditional tester skills. Strong attention to detail. The ability to think outside the box. To be able to spot a potential bug or defect from a mile away. To write a great defect report with detailed recreation steps. All that good stuff. *But there is so much more as well*.

You must be able to read and understand the development code. I mean **REALLY** understand it. To a point where you can explain why something is wrong or inefficient (and how to improve it).

You also might need get involved and write your own development code. I am noticing more SDETs getting asked to write development code. This doesn't have to be a bad thing, far from it. Although it is yet another skill required to succeed.

On top of that, you will need to write and maintain the **automation testing framework**. Write the code for the tests. Implement them into the **CI** system. Determine where the **mocks** are required. Write them and put them in place. Write the **UI tests**. Write the **Integration tests**. Write the **End to End tests**. Write the **Unit tests** (if you are really unlucky…).

Does your application need **performance testing**? Of course it does! So find a tool that you can use for performance testing (and forget about budget, *open source only…*). Write the tests. Create and organise the test data. Procure the test environment. Of course, it needs to be dedicated. It also needs to be a replica of production…. Good luck with that in most organisations… Arrange the monitoring and instrumentation. Execute the tests. Analyse the results. And when (not if) it all falls over, tell the team exactly where the problem is.

Of course there is also the need for **test strategy** and **planning documentation**. OK, with the rise of Agile there is less documentation required these days (thankfully). *But these core documents are still required*, and guess who will be writing them? If you don't write them, people will definitely ask why not. If you do write them, no one will read them anyway. At least not until something goes wrong.

I'm going off on a bit of a ranty tangent here. But hopefully you get my point, this is a lot of skills required to succeed.

Reason 2: You need to have a multiple personality disorder

OK, perhaps I am exaggerating here a touch…. *But you can't just think like a tester*. You need to be able to think (and act) like a **developer**. To understand why they have done things in a certain way, and why that makes sense (or not).

You also need to be able to think like an **end-user**. *There isn't just one type of end-user either*. Some are smart and know what they are doing. Some less so. You need to be able to think like them both, and everyone else between.

Then there is the **product owner** and **stakeholders**. What would they think of this system? What are they likely to ask for next? Would they be happy with how this has been implemented?

This is a lot of different caps. *You better get used to wearing them all*.

Reason 3: You need to be good at delivering "Bad News"

Everyone loves delivering (and receiving) bad news, right? What could be better than going over to the downtrodden, emotionally unstable developer's desk. For the 50th time today. And telling them that the code they have poured their soul into still doesn't work. *What could be better than that… ?*

Any tester will have felt varying degrees of **being seen as an inconvenience**. Or as someone who "likes to cause problems". *Ironically, the better you are at testing the more you will feel like this*. **Bugs are often seen as a fault rather than help**, even though that notion is ridiculous. You don't want to fix these bugs in production, do you?

How about presenting the latest performance test results to the leadership team? The results are shockingly bad, and going to cost a lot of time and money to fix. There is no way you can launch next month as planned. It's not your fault (at least not entirely, you are one of a team, remember?). **But it's you standing there delivering the news, sweating like Josef Fritzl on MTV Cribs**.

Reason 4: There is no training

Alright, that isn't *strictly* true. Of course there are training courses available such as the ISTQB certifications. But I think they have questionable real world value due to their subjective nature. It can be difficult to apply the theory you learn into practice in reality. *Experience is key to software testing*. There is no substitute for it. I found that this post on the value of ISTQB certification from Simon Prior shared my views. He summarises by saying that the knowledge needs to be applied extensively in the real world.

Let's compare this to development for a moment. Experience is without doubt important here as well. But say you know that you are going to be working on a certain technology stack in your next project. You can take *specialised and relevant* training courses on that technology. You can study other materials and example code on those technologies. Then you put what you have learnt into practice. If you run into difficultly, you can post the specific problem to a forum. It often isn't that straightforward for testing.

Reason 5: Recruitment is a nightmare

Have you or your organisation had to recruit anyone in testing recently? How was the process for you? I have been involved with or conducted the interviews of over 100 testers in the past few years. *It has been a difficult and frustrating process to say the least*.

tend to list a huge array of skills and technologies on their CV. When they are pressed on any of them, they struggle to answer even basic questions.

The problem is that it can be hard to assess and judge the skills of a tester in a couple of short interviews. You can query them on testing technologies and various theories. But answers to these questions can be quite easily learned and recited.

A common problem I find, that is true of both candidates and recruiters, is **too much focus on Testing Tools, rather than core Testing Skills**. James Pulley touched on that in an interview with Northway Solutions, and it is often discussed on the fantastic PerfBytes pod cast (if you have any interest in performance testing, it's well worth checkout out).

Perhaps a better way is to get them to talk through how they might test an application with components A B and C. What testing would they instigate, and *what issues would like look out for*? This still doesn't ascertain whether they could implement what is discussed, but it would be a start.

Development isn't immune to these issues either. Goodness knows there are enough poor quality candidates in the field as well. But a developer can be set a *specific coding challenge* relative to the job requirements. Suppose they can pass that, and answer some suitable technical questions on WHY they did what they did. The interviewer has something far more concrete to go on here.

Reason 6: Less Involvement, More Expectation

documented. Every test case should have been executed, even if the testing window was only 1 day because development overran (again…). The expectation is that testers are certain there are no bugs in the system. *Everything should work perfectly when moved to production*.

All this expectation often comes with less involvement. Testers often aren't queried on what THEIR requirements are for the testing. *Whether what has been developed is even testable*. Or how they will complete the testing. Or how much time they need (which can be difficult to quantify anyway)

When things do go wrong in production, *t* **te** **Keeper of Quality**", so why wasn't this p or this (complex and almost impossible to rep

Thankfully, and again with the rise of Agile t is more widely accepted that the whole tean Masset, my Director at Concur / SAP, recei

Testers should no longer be treated like **se** organisations. If you are still treated like o ge them (or they won't change them), **find a** *professionals is at an all time high*.

Reason 7: Less credit and appreciation

Again, this reason is less true where developer and tester are seen as equal (or close to it, at least). But elsewhere, the common scenario when everything works well is – "*The Devs did a*

This doesn't bother me much on a personal level. I have learnt to award myself Gold Stars, rather than wait for someone else to do so (thanks in part to this great book – The Success Principles). But of course, it never hurts to get recognition for a job well done. *This can be hard to come by for testers*.

Reason 8: Y

Maybe this is a bad thing, maybe not, you ers in most Agile teams seems to be 2:1. Perh e expectations placed on the tester.

The expectation might be to pair with the type of test (unit, integration, e2e). And get all testing. And write up all the test results an ... developers only drink coffee). That's a lot

Being outnumbered naturally means that testing has less of a voice. *That doesn't need to be a problem though*, **we just need to shout twice as loud**.

So there you have it, its official, testing is harder than development! Well, maybe it is, maybe it isn't. But I think with the points I have outlined above, there is at least some justification to say that.

Tags: Development QA Testing