Gojko Adzic to me Jan 24

Alice in the hall of mirrors

Why teams often fail to get the full value out of Scrum, Kanban and TDD, and what to do about it.

The mind of the nineteenth century English writer Charles Lutwidge Dodgson was a curious mix of faith, art and logic. Coming from a long line of English clergymen, Charles was educated from an early age to join the Church of England ministry, and was even ordained as a deacon in 1861. His interests, however, were not really theologic. During his twenties, Dodgson had a keen interest in the emerging art of photography, and became a famous 'gentleman-photographer'. Despite not having access to instagram or cloud storage, Dodgson took more than three thousand images before his photography interests waned. He then went on to become an accomplished mathematician, publishing many works on symbolic logic and algebraic geometry. It's no wonder that such a wide variety of interests brought wonderful imagination, captured best in his more popular works under the alias Lewis Carroll. His literary works are a unique mix of wonderful imaginary worlds, witty dialogue and logical nonsense. One of Carroll's most recognised stories is the fantastic journey of Alice through a 'looking-glass', inspiring generations of children to think that there is a whole world on the other side of mirrors. As if struck by Carroll's wonderful storytelling, many grownups today still expect mirrors to somehow magically open and reveal their secrets, especially those working in the software industry.

The world of software is full of mirrors. Scrum boards show where the work piles up during an iteration. Kanban swimlanes expose queues and buffers in a process. Value stream maps shine light on hidden constraints and bottlenecks in organisations. Retrospectives expose systemic organisational blockers that slow down software delivery. Context maps show the interactions and dependencies between teams working on the same piece of software. Build monitors show whether the components can be integrated and deployed. Lean startup advocates relentlessly measuring effects of our work so we can make better informed product decisions. These tools show our reality back to us, so we can make better decisions. They are just mirrors.

On one hand this is perfectly logical. Software delivery is so contextual that it's difficult, if not impossible, to prescribe context-free practices that can be successfully applied to various teams. Rather than offering advice on what to do, widely applicable tools tend to focus on making people better aware of their own contexts. On the other hand, this situation can be incredibly frustrating for people new to these tools. Mirrors don't really offer immediate actionable advice on how to improve the situation. Without knowing where to go, the feedback provided by techniques that show us back our context is almost useless. This is nicely illustrated by Lewis Carroll in the famous logic twist when Alice came to a fork in the road and saw the Cheshire cat in a tree:

'Which road do I take?' she asked. '
Where do you want to go?' was his response.

'I don't know,' Alice answered.
'Then,' said the cat, 'it doesn't matter.'

Mirrors, both virtual and real ones, have two key aspects that we need to understand better in order to benefit from them, and avoid wasting time.

The first important aspect is that mirrors are captivating for a short period of time. The centre-piece of Louis XIV's Palace of Versailles was the Hall of Mirrors, designed to awe visitors and residents alike. Software teams similarly get mesmerised by their own mirrors. As the lean startup approach gained popularity, wonderful dashboards popped up on large screens in developer rooms all over the world. Those screens are hypnotic, and the very fact that the numbers on them are constantly moving can provide a false sense of security that we're making progress. Yet few teams know how to actually conclude something useful from all their user metrics, so the difficult process advocated by Eric Ries frequently gets shortened to something much easier: build and measure, don't bother with learning. After a while, the user engagement monitors get repurposed for something more immediately useful, such as build statuses and database uptime metrics.

The second important aspect of mirrors is that they are a cold, emotionless reflection of the current situation. Organisations new to Scrum often complain that they've been trying cork-boards, retrospectives and stand-ups for a few months, but the problems don't go away. In fact, they're piling up. Real-world mirrors are horribly boring compared to Carroll's fantastic looking glass, or the Sun King's extravagant hallways. The good ones only show the reality, undistorted, and often not convenient. Seeing something bad staring back at you from the mirror is frightening, and no matter how much we hope for it, the problem won't go away just from looking at a mirror more frequently. Similarly, seeing something bad staring back at you from a scrum board, a value stream map, or a Kanban queue, doesn't improve the situation in any possible way. Without the context how to fix the bottlenecks, a Kanban board just makes people depressed.

This might sound like common-sense, but a recent big realisation for me was how there is a huge gap between two groups created by these mirror-techniques. People with enough context to take positive steps often see mirrors as an actionable tool, not just a feedback mechanism. People without enough experience to get good conclusions expect detailed actionable advice, and after hypnotic effect of a mirror wears off, give up and discard those techniques as useless. And understanding when we're dealing with a mirror is critically important for making those techniques work and avoiding silly arguments.

This is perhaps the most obvious in the discussions about TDD, especially the recurring theme that TDD can lead to crap code and bad architecture. On one side of the gap are people who consider TDD a design technique, and mention it in the context of tests driving towards good design. On the other side of that crevasse are people who complain that they've written hundreds of unit tests and still ended up with a bad design that's difficult to maintain. On the third end of the crack are prominent members of the developer community, such as David Heinemeier Hansson, or respected authors such as Jim Coplien, who mostly consider TDD as a waste of time. And all three groups are quite dismissive of each-other.

The situation is easier to understand if we consider that, in the context of TDD, the unit tests are also a mirror. TDD just makes it painfully obvious that a design choice is bad because tests hurt. Methods with to much responsibility hurt to test. Objects with too many dependencies hurt to test. Components that are too tightly integrated hurt to test. But people still need experience and design skills to act on those signals. The irony is that for people with enough design skills to take the corrective action, TDD does lead to somewhere good. But people without that context just experience the pain. TDD does not directly lead to good design, but it can lead away from bad design if people know how to listen to their tests. And of course, it's not the only mirror that provides signals about design. Experienced developers with a different technique can also achieve good design, and may not need that aspect of TDD at all.

On one hand, bad experience and lack of actionable guidelines make it difficult for less experienced people to improve. On the other hand, not being able to separate mirrors from actions makes more experienced people seem too dogmatic and disconnected from ‘the real world’.

If you’re reading this and it seems to you that some seemingly amazing software practice just doesn’t help you improve anything, consider perhaps that you’re dealing with a mirror, and that it’s not designed to actually provide direct actionable steps. Perhaps seek out someone outside your context to provide suggestions what to do about it. If you’ve collected a ton of user metrics but don’t make any decisions based on them, or even worse you’re making bad decisions, don’t give up on lean startup yet — perhaps hire an experienced product manager who can suggest what to measure instead. If the testing queue is filling up on the kanban board despite all the stickies you’ve bought, don’t dismiss the whole approach yet — perhaps post a few messages to online testing discussion groups and explain the context, more experienced people will suggest some actionable ideas to try.

If you’re on the other side of this looking glass, and your colleagues do not share your enthusiasm for something you know works well, think if it’s actually a mirror as well. Consider that it’s useful to mentally divide the feedback tool and the activities that lead from it. The more experience you have with a feedback mechanism, the more interesting for you it will be to talk about different ways of setting up mirrors and exposing the context. But people you work with, or people you’re speaking to at a conference, might need actionable advice for dealing with feedback instead of learning how to gather that feedback better. If your colleagues are reluctant to try out stuff you’ve seen work in other places with great success, work with them on improving the process based on what the mirror shows, instead of explaining how the mirror works. Give them practical advice in order to build up enough context to act on these signals on their own.

As for TDD and tests, my pet peeve, here’s an experiment: I’ve created a GitHub repository for people that need some clear actionable advice on dealing with tests that hurt. If you’ve seen enough ugliness in a mirror and don’t know what to do about it, create a new issue there, post some code snippets and explain the context, I’ll do my best to offer some clear actionable advice. For anyone else reading this, if you’ve gone through the TDD journey enough to know how to act on hurtful test signals, please join in and review the issues as well.

Photo by Matteo Vistocco on Unsplash

- Tallinn, EE, 3 October

Getting Started with Serverless Architecture

- London, UK, 20-21 February
- Oslo, NO, 16-17 April
- Budapest, HU, 9 May (Craft conference)

If none of these dates are convenient for you, get in touch to organise an on-site workshop for your team.
