Generalised Agile is a Pipe-Dream for Most

Martin Hunter, August 2018
https://www.continuity.kiwi.nz

Countless articles have been written lately that discuss how to do agile 'right' and how to do it 'wrong' in the context of generalised usage, a way of running your business. The conclusion is often that, where failure has occurred (which is often fairly catastrophic – no small irony given agile's 'fail fast' principle) the diagnosis is that the organisation in question didn't align itself properly with agile (often managerially) and the treatment is to drink more agile cool-aide. The often quoted example of success is Spotify – the online music streaming service company (a firm, incidentally, whose core business is software). The often quoted examples of doom are usually institutionalised government and commercial entities (core business: not software). To me, it seems fairly obvious that the context, structures and incentives of a online music streaming service would be vastly different to those of a centuries-old wing of democratic government that plays a pivotal role in the health and well-being of a nation, so I'm not sure of the value of the comparison. But I'm a technologist, I am open to new ideas and I don't dismiss new proposals. I am also aware of the Gardner "hype curve" and as such usually remain sceptical about new ideas until they have proven themselves.

The article got me thinking – in what sorts of organisations would generalised agile really work? Large, institutionalised organisations are often the ones being held up as the anathema to agile; the troublesome, burdened shrines to heavy-weight planning and bureaucracy that agile evangelists like to challenge and dismiss. It hardly seems necessary to point out that such organisations have trouble implementing generalised agile, though I do find the claim that generalised agile would work in such organisations necessary to challenge – after all, it must be the case that such organisations were not always institutional and planned and have evolved to become like that for a reason. What about the other end of the scale – entrepreneurs? Small- and sometimes medium-sized businesses that thrive on speed-to-market and innovative product ideas? I imagine myself trying to explain generalised agile to a "Dragon's Den" of venture capitalists. The story would go like this: “Welcome, so what’s your pitch?” they would ask. “Well,” I’d reply “we’ve got a vague idea. What I’d like is for you to give us a bunch of money, and we’ll start with something small and see where that takes us. At the end of it you’ll get something.” There would be a slight pause as the Dragon’s Den tried to figure out whether I was serious or not. “Ok, so what’s your plan?” they’d ask. “Well, we don’t have one beyond the next 4 weeks. But we’ll keep going until the money runs out.” “So what exactly are you going to do?” they might ask, beginning to look bored. “We’re just going to keep listening to customers and let them drive our decision-making.” At this point, I do believe a Security guard might start to usher me away from the microphone.

Before I come across as anti-agile, I will say that Spotify is a great example of how to do agile right. But it is important not to loose sight of the fact that it is also a great example of the sort of context in which agile works well. You can’t presume agile to work well in all contexts, just because it did on one or two. “One sausage a BBQ does not make” as a good friend of mine is apt to say. How many Spotifys came along and failed before Spotify? Were they all agile? Had they all drunk from the fountain? Was their agility the sole reason for their success or failure?

Like always, the problem here is not in the methodology or technology, but in expectations of the applicability of those and the human tendency for wishful thinking. Agile sounds like a good idea. After all, who doesn’t want to be agile? But the key principle behind agile is about being able to adapt to changing needs and direction. This presupposes several things — that needs are changing, that they are changing fast enough to make agile methodology the most effective way to go, and there is no inherent advantage in actively inhibiting change or even simple procrastination (Procrastination, I’ll point out, is seen by the risk specialist Nicholas Taleb as a form of systematic “redundancy” that is very useful with regards to successfully responding to high-impact, low-probability, difficult-to-predict events that he calls “Black Swans” — a sort of event that agile also aims to manage.) The reality is of course that nothing is so simple. As I mentioned above, human societies are littered with organisations that have evolved to become institutions. Sure, they may not respond quickly in every - or even any - situation, but maybe that’s because that’s what society wants from them. I would agree that it’s highly likely that most institutional organisations would benefit from agile in part, but I very much doubt the general applicability of agile to such organisations. The same goes for entrepreneurs — yes, they're cutting-edge, risk taking and demand fast turnarounds on products to beat the market. But that does not come at the price of good business sense which includes a multitude of careful research, planning and projections, and I cannot imagine a successful entrepreneur managing their affairs otherwise unless they are just lucky.

So, if generalised agile is applicable only to a small number of marginal, dare-I-say "freak" organisations then what is agile good for? Well, I'd like to draw us back now to where agile came from: Software engineering. This, in all its forms, is an enormously complex endeavour. It doesn't matter whether you're coding or configuring, customising or integrating, software is a very tricky craft. It is tricky not because computers or electronics are complicated (which they are) but because — to quote J.R.R. Tolkien's Gandalf character — "not even the very wise can see all ends". People who want to use some software generally don't have a very precise idea about what they want (well, not precise enough to write or configure software from anyway). This itself leads to two general modes of operation for those building software — one, create something and hope hope users like it, or two, figure out what users want in more detail and build it. In both cases, responsiveness to changing requirements is important. In the first case you get changing customer feedback, changing business priorities from your own organisation and people in your team coming up with new ideas that you want to implement in your software. In the second case, it is almost always impossible to specify software precisely ahead of building it and get it right, and also people change their minds, often once they've seen something in action, and you want to be able to adapt. In both cases - owing to the underlying complexity of the many layers of rules that comprise software systems - software bugs, large and small, always turn up which will impact you and drive a need to change. The uncertainty inherent in software engineering, and the need to adapt to changing certainties, impacts your software in two main ways: the way it is structured (it's "architecture") and the way you manage changes to it (what we call "software delivery lifecycle" management).

Extreme advocates of agile often assert that "architecture" as a practice is not necessary. This depends a lot obviously on what you mean by "architecture", but overall I don't think this is the case – firstly, all software systems have an architecture, whether you are aware of it or not, and it is useful to know what it is in order to verify its applicability and flaws; and secondly, agile practice in and of itself does not address structural architectural concerns very effectively at all. Imagine you are building a road. You know that a road is made of various components and built in a certain way – so you are not completely ignorant. The minimum product is a road that goes from A to B, and you start building it. About mid-way between A and B you run into some trouble – the ground is very soft. Hang on, it's actually a shallow lake – no hang on, it's a swamp. What are our options? We can't turn back because we've built half the road (sunk costs), and doing so would mean having to reverse up our trucks which will cost lots of money now that they're perched precariously on the edge of a watery grave. We have to shimmy some sort of bridge in. Ok, let's do that. But we don't have the budget for a proper bridge so we'll hack something together out of some left over concrete piles and some timber that was used for packaging the port-a-loos when they were delivered. Finally, we get across. Meanwhile, there's a person standing on the sideline looking at what we're doing. Earlier they had stood back a bit and saw the swamp using some binoculars. Through a bit of planning, some research and field measurements they'd drawn up a proposal to route the road around the swamp. Sadly, the proposal started out going in what appeared to be the wrong direction and the ignorance of our team was alas masked by our enthusiasm, so those who make the decisions didn't take enough interest in the person's plans. Now, we're stuck, because not only have we just spent a more vast quantity of money building an unnecessary rickety bridge over a swamp we didn't to look for but was clearly there if we'd used a pair of binoculars, but all the traffic is down to one lane and can only do 30kph over the bridge. This is not to say that agile principles or methods are wrong – far from it. Ability to respond to change is useful. But it cannot come at the expense of planning enough (both business and IT wise) to see the swamp – the context of our endeavours, the details of them enough to be confident that we're on the right track (or road). As we have found throughout the history of IT (and humanity for that matter), it is sensible and prudent to do some research and put some thought into what you are going to do before you do it. This requires a bit of analysis, a bit of architecture and design, a bit of resource planning and a lot of expectations management. A further reason that "architecture" is not going to be superseded by agile overall draws back to my first point — that all software systems have an architecture, whether you are aware of it or not. The job of certain types of architects (software architects, often called solution architects) is to know and control what that architecture is and to balance decision making about its structure across multiple competing stakeholders needs. To paraphrase Winston Churchill, not all stakeholders can be pleased all of the time. Agile software teams without architects rely heavily on product owners (often non-technical and ill-equipped to spot and evaluate trade-offs involving technology quickly) to manage stakeholders expectations, and on programmers (often non-business minded and ill-equipped to spot and evaluate stakeholders needs) to make technology choices. This is why agile team members must be "alphas" — really smart, hard-working, productive and highly experienced. It goes without saying that these people are like hens teeth. In addition, the dynamics of the IT industry over the last 30 years (read: if you want to progress your career and earn more money as your experience increases, you can't stay in programming) has lead most of these people into roles such as software architects, business analysts and project management and away from full-time programming. Throwing away such people therefore throws away the very people you need on your agile team to make it successful. Agile I think asks for such people to return to programming and testing, at least in part: I get this, I think it's a good idea. I think software architecture should be more demonstrative — show programmers what to do through examples, chip-in to writing production code when required, make regular use of "spikes" (prototypes) written by the architect to mitigate risks. The same is true of analysts and project managers. But you can't throw them out and expect a better outcome.

What we don't want — and I see everywhere — are conversations that go along the lines of "trust us, we know what we're doing" without offering hardly a valid platform from which to make this claim. This is pure doublethink: You can't possibly know what you're doing if you haven't planned adequately, to claim mutual truth in these contradictory positions is nonsense. The roles and responsibilities that agile proposes have not been established over centuries, with the wisdom of generations of experience and governed by industry charters and statute, it comes from software engineering which is barely 60 years old. And even in software engineering it works in some contexts, certainly not all (I refer you to

Gartner's concepts of Pace Layering and Bi-Modal IT Management). The history books are littered with failures of all kinds of methodologies in IT, why would an unproven propagation of such methods to more general business be more trustworthy than the principles upon which business has been founded for thousands of years?

My message here is not to be a neigh-sayer but to be a realist. Agile is good at some things in some contexts. It is not a one-size-fits-all answer to organisational troubles or to business success.