4.

DECISIONS

One of the hardest things to do is to explain decisions. Specially when the mood is that "anything coming from outside of the team is a Top Down".

Top downs are sort of a silly ghost haunting teams, which are hard to chase. They don't exist most of the time - what exists is an asymmetry of the decisions based on accountability, priority, urgency, execution power or capabilities.

You'll see smaller teams taking over old corporate IT structures to build instead of buying some of the components of the business. You might be the one leading it, the victim in the losing side of it or just a passerby. Depending on it a decision of one team can be perceived as causing a Top Down to the other.

The contrary movement happens too: tech as commodity — when all technology you have is from outside vendors. It comes as "We have the brains, they execute" or "you guys spent 2 years trying to reinvent the wheel and we're making an Executive decision of buying Big O software". Same energy and awareness issue. Where transparency failed ? Where all nice agile presentations were gone ?

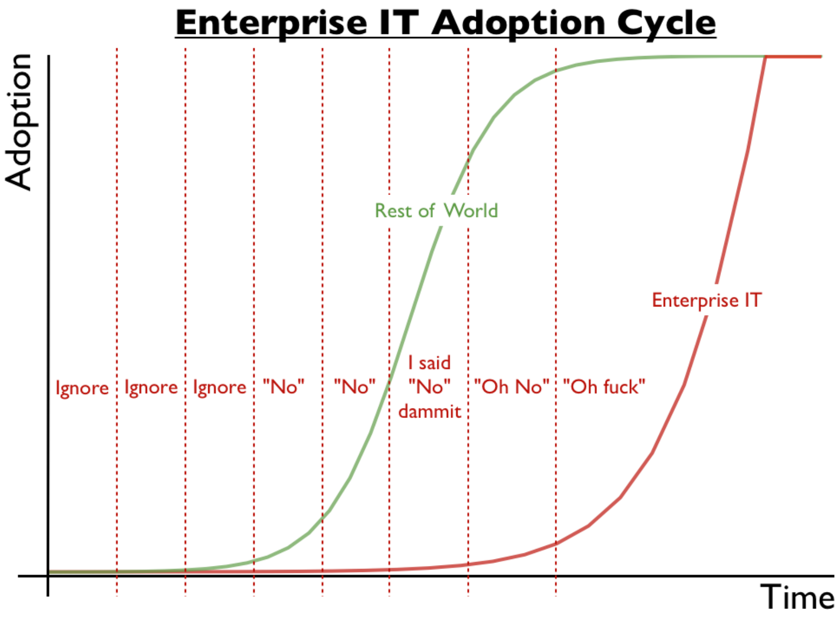
In both cases proper planning, objectives and how to measure success is left in second or third place in lieu of knee-jerk reactions. We will see that below, after The Plan.

The importance of knowing what your team does as we discussed in the last chapter, what your company do and having the courage to express your point of view is that you will be closer to the birth of these decisions.

They are not made in the cafeteria or in all hands gathering. But having your leader genuinely part of such decisions make it easy for a team to deal with them. No one feels comfortable with a leader that is a victim all the time.

Top down decisions are result of storming out frustration and discussions, which if you can't negotiate your way, there is a potential of alienate your tech team.

In all cases the side effect is people leaving because if the higher ups don't care, why should they ? (remember they are not afraid of losing their job or not finding a new one). So being close and managing them - better yet avoiding the point where they are made - is good for your team.



"ADJUSTING" THE CULTURE

All that said, while I was writing this post's drafts I stumbled in some news that caught my attention in companies with strong engineering culture and that I see as an adjustment of culture in face of results they were getting.

On real big companies this is as hard to see from outside as finding planets around distant stars. We rely on leaks, investors reports, outages and mass firings to get the data. Ah, and by "adjustment of culture" I mean top-downs.

Facebook motto was "**Move fast and break things**", a motivational piece to own mistakes and quickly move forward. At the 2014 F8 event, they changed it to “**Move Fast with Stable Infra**”. The reasons behind it may not be different from other companies: investors, regulations, predictability, MTTR, SLAs and better customer experience.

Yahoo published an analysis that by cutting out QA steps led to better quality, shorter delivery times and other goodies. It does not mean that they are throwing code to production without ensuring that automation visibility is in place. It means the accountability shifted from a QA team to the engineers, using a continuous delivery system, when the code goes live if anything breaks it must to be fixed within the proper SLA. No more CYA by complaining about QA.

A couple of years ago a former Amazon engineer published the "Jeff Bezos Mandate". It is a commentary about Amazon being a platform and a list of things mandated and enforced at the engineering level to ensure this platform was to be:

1. *All teams will henceforth expose their data and functionality through service interfaces.*
2. *Teams must communicate with each other through these interfaces.*
3. *There will be no other form of interprocess communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network.*
4. *It doesn't matter what technology they use. HTTP, CORBA, Pubsub, custom protocols — doesn't matter. Bezos doesn't care.*
5. *All service interfaces, without exception, must be designed from the ground up to be externalizable. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions.*
6. *Anyone who doesn't do this will be fired.*

It doesn't seems important, but number 3 alone would warrant weeks of meetings at some companies and a round of beer on others.

And by the rest of the post looks like Amazon enforcement was a bit more stronger than a weekly passive/aggressive reminder.

What we have there:

- Going from crazy to crazy with stability is required, find your synchronism and don't bother the business because you wanted to push linux as your main network router and it failed.

- Cutting out the test pipeline and telling engineers to own their quality, not because QA is bad but because cutting out who to blame forces empowerment. Saying you haven't delivered because you didn't have a free QA or DBA is not acceptable.

- Jerry-rigging your stuff on any place is enforced with a ban from the paradise. This basically cuts any kind of slack you give to people that can't design and run their code alone but can't make the time to work in a team context. Business don’t know it until they know the side effects.

Look at the companies I'm talking about: Facebook, Yahoo and Amazon. These adjustments on culture and behaviour means that a set of practices and culture paid off in a way that the traditional processes surrounding IT didn't.

These processes mean less to leadership and execs than it does on other kind of companies. The team owns what is in production respecting a certain set of rules or a limited freedom. Ownership, accountability and other nice words are used to describe this effect.

But before moving to processes, I want to talk about another euphemism for top downs:

DISAGREE AND COMMIT

Disagree and commit means more than what is discussed on modern self-managed teams or management 3.0 material. Company performance awareness is important during growth.

As we saw in the adjustments from Yahoo, Amazon and Facebook that we could track, there might be top downs at the executive level but more often disagreements will happen in local groups.

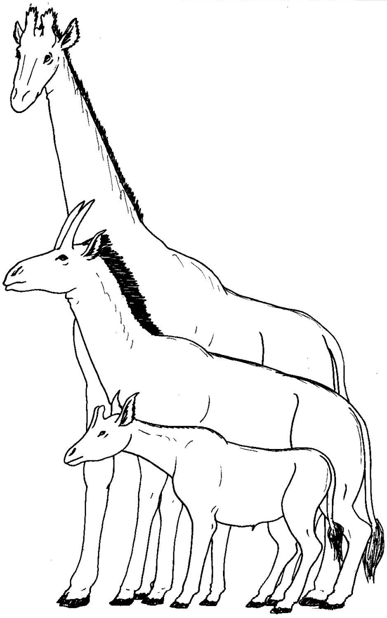
Ideally discussions would not extend beyond what the culture support. Some people will be more laid back, others more vocal. Once in a while, you will have an argument but usually it settles back.
When tech leadership is missing, there will be long discussions.

It is useful to me to track if no action is taken while the discussion volume increases. If no action is taken, no one moves, it quickly becomes meaningless. It becomes an ego exercise as I mentioned before for bikeshedding.

If no action is taken, if the team is constantly discussing it will impacts delivery.

The company will look for tools to help unlocking that from a high executive perspective: institute devices as the ARB — **Architectural Review Board**.

In an attempt of not telling people what to do or owning up decisions you end up with a tribune to replicate the same discussions in public, like old Rome. The other common device is bringing in outside experts.



A horse designed by committee

These are the answers of busy executives mixed with the generic "communication problem". There will be no meaningful decisions, at most the most vocal person may get some space but this kind of personality rarely comes with good execution skills. It signal strongly that there is no effective engineering leadership.

It feels different than *"disagree and commit"*, it is a *"sit down and be quiet"*. And far from *buy-in*, the side that "lost" (as they will rightfully perceive the outcome) will just wait to get delighted by the failure or collectively think about leaving to create an impact.

This may cause middle manager leadership to leave, which is a team as hard to build as a good engineering team. Naturally we think that if the not-so-good people leave we will be in a good spot but that's rarely the case alone.

There will be different composition to the horde of unhappy, and there is a cascading effect coming from the conversation this team have out of band. It is more common that competent people leaves altogether.

PROCESSES (OR WHY I LEARNED TO OWN PRODUCTION)

Another "disagree and commit" point are processes. Traditional companies resort to ITIL a traditional IT methodologies in hope that it helps figure out operations and delivery. When it gets to development, the same formula is applied with Agile methodologies or equivalent factory based ideas.

In both Development and Operations cases teams are assembled around these processes and meetings scheduled to link them to existing processes, converging to a spreadsheet and someone asking why deadlines are not being respected or how can we shrink the budget. Corporate managers versus Technical Managers.

A devops clock

At the same time in the last years DevOps, Facebook's Production Engineering and Google's SRE got in the middle of both teams, sometimes not to substitute them but as cry for help from development teams for more speed and freedom.

Google released a book on their SRE practices, which describes how they deliver service and software in a reliable manner. Companies tries to adopt this bundle of mixed methods to move fast but at the same time protect their business. In the sections below you will see some of these practices selected to help you quickly get on your feet.

I don’t think development and operations are not so distinct that they need different people and skillsets. I like to call this set of activities Delivery, which means the whole. Some business verticals and government rules require these activities to be separated (separation of duties) to allow your company to operate or to buy from you but that's not an absolute role.

The dev team fixed the devops clock

On the other hand, people with limited experience and weak self assessment fails miserably when trusting the same things it takes to build CRUD apps applies to operations and the reverse: when thinking that by knowing Ansible and Python you can build business applications right away.

What most traditional companies have not explored is taking out these processes-based roles, and probably people that fill these roles, out of the production pipeline. This is what companies like Facebook and Yahoo are saying they are doing when they say they shortened the path for production. In shorter words — distribute ownership. That is usually named ***"You build, you run"***.

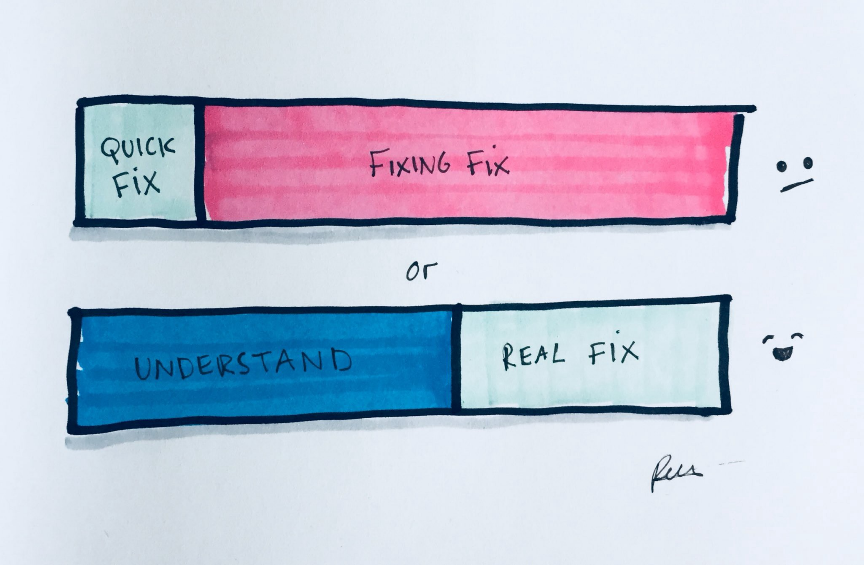
The regular production pipeline of creating software comes from the company/market/sales needs and ideas. It goes through a hopefully incremental development and deployment cycle and lives in an environment that is taken care of.

Customers use the new product, ask for new features, complain when it is offline or broken, the company lives through its pain, people exchange email blaming each other because things are not perfect and life goes on asking permission on processes that should ensure quality and stability.

In the book "High Output Management", Andy Groove (Former CEO of Intel) uses a breakfast factory to illustrate production line among other things. One of the key metrics to plan a proper production line is the time on the task that takes more time. It is an interesting book to explore as it embodies most of how we work today.

Many scenarios are explored and it is possible to grasp the idea of waste, delivery time and investment on production. In many cases, tentative improvements to machinery in a single place of the production line come out as waste when put together at the end. I can't recommend this book enough as it is an historical document where you can see the beginnings of many of our industries practices there — for example OKRs.

Your job requires a set of skills and ownership hard to achieve first hand so empathy can be used to understand what is going on when your organization has this division. If your tech org still has functional team it means that for whoever that is accountable in a higher level things are not as green as it appears. Be sure you don't know everything about all the things.



shamelessly stolen from https://twitter.com/chopeh/status/926074073767206912