Product and Engineering Delivery

90-day field guide

Intro

Who am I

UOL, Terra, AT&T, Locaweb, Luizalabs, Lucid, NUBANK,.

Team builder, engineer

Not a CIO.

Managed orgs up to 270 people, company wide processes for +500 engineering teams (perf evaluation, large scale migrations, budget, break even, pré-IPO roadshows)

Business awareness through product management

Why a field guide ?

I've been through a couple of companies in the recent past,ranging from more traditional to tech startups.

I tried to summarize what I've learned to help tech leads, engineering managers, directors and CTOs.

I've had good and rough starts (same for endings), but I also had great teams and projects that makes me proud.

All in all, this made me think that we work with people and that tech is a side effect.

If we are mindful of some important things while we join a new team, our life will be simpler and easier. And as a field guide goes, this is what I do to make my life simpler.

The first 90 days

Concerns and what to look for

Measure

Challenges

Team structure

How delivery happens

Closing advices

Measure

Engineering Delivery Metrics

How to baseline delivery across diff orgs

- ➔ Deployment Frequency
- ➔ Lead time for changes
- ➔ MTTR
- ➔ Change Rate Failure
- ➔ Incidents
- ➔ Cloud Economics (monthly/yearly cloud expenses, licenses costs, support and enterprise deals)

Team Metrics

What is a good team ? Where does the work flows from (and to)

- How the current organization work ?(teams, squads or tribes)
- How many unhealthy teams: no manager, stretched managers, no product manager
- Work distribution issues: work that should be done elsewhere, duplicated work, prioritisation
- What is the priority definition between engineering and products ?
- What is the decision making process (RFC, committee, go horse ?)
- Are there clear levelling for ICs and Managers ?

HR and Engineering Org insights

Grey areas, risks and what's everyone thinking about

- ➔ How is hiring organized ? Is everyone helping ? What are the goals ?
- ➔ Are functions hidden (infosec done by a SRE, prioritisation done by committees) ?
- ➔ Any functions dependent on a single individual ? (e.g. the person that knows legacy code or how to deploy, or point of contact to solve an issue)
- ➔ Any team missing ? Infosec, SRE or Platform infrastructure, Engineering tools, Data engineering, Data Science, Product Engineering, Digital channels, Notifications teams
- ➔ Is remote work allowed ? Is it successful ?

Operational Qs

How does tech fits in the company operation ?

- ➔ Is there an incident process ?
- ➔ Is there an incident severity matrix, blameless postmortems and product feedback ?
- ➔ How is productivity measured ? Any product vs engineering stalematches ?
- ➔ How are incentives aligned (rewrites vs new features, incidents vs growth)

Challenges

Challenges

Joining an existing team is hard, specially if it is a reorg. Where will rejection come from ? What failure looks like ? Where are the leverages ?

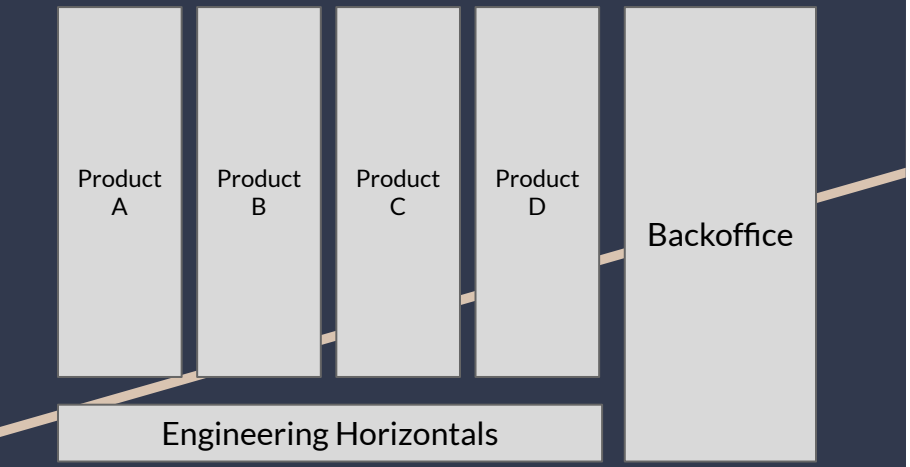
- ➔ How they will react to a new structure or a new high level executive ?
- ➔ What "Lead by example" means in the company's context ?
- ➔ Which public forums the team have ? How they can be heard ? (all hands, town halls, teams gathering)
- ➔ Unclear dotted lines: teams that have no clear boundaries.

Team structure

Companies nowadays try to adopt the Spotify model

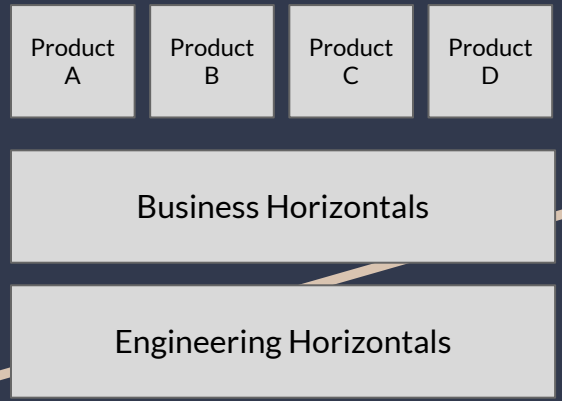
- ➔ Multidisciplinary tribes, layered teams (each and every team requires individual of all chapters: product, eng, data, ops, finance)
- ➔ Pros: one stop shop for business verticals, quick reaction time to day to day business needs
- ➔ Cons: amplify the cost of simple decisions, as business grows it requires more high level arbitration
- ➔ Can't attend strict business or engineering tasks i.e. finance, infrastructure as the goals can not always be consensus driven (costs, undifferentiated lifting, standardisation)
- ➔ Legacy product and code management is hard

But the tribe model is usually too stretched



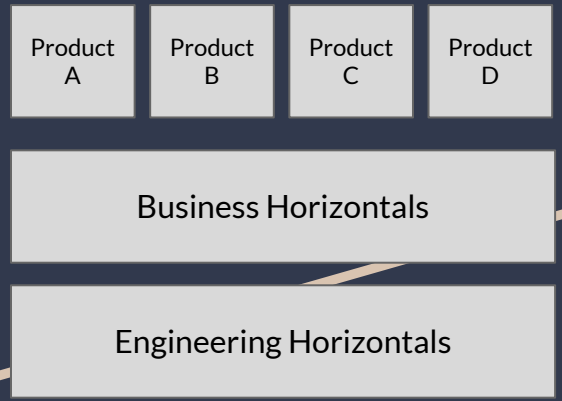
- ➔ Cross business concerns are spread across individuals
- ➔ Hard to identify duplicated decisions or structures
- ➔ Shared efforts are hard to push - they compete with local priorities

Shorter tribes with focus on delivery



- ➔ Keep tribes where it make sense
- ➔ Invest on platforms, bottom heavy, light on product/customer interfaces
- ➔ Identify shared efforts
- ➔ Speak product language even on internal customers
- ➔ Shared P&L

The layered model benefits engineering



- ➔ Nimbler product teams
- ➔ Foster a strong dependency on APIs
- ➔ Business horizontals: Auth, APIs, Infosec, Notification, Digital channels, Design, Finance
- ➔ Undifferentiated lifting done once and for all
- ➔ Eng horizontals: Data, Infra, SRE, Tooling

How Delivery Happens

The four Accelerate metrics (2017)

	High Performers	Medium Performers	Low Performers
Deployment Frequency	Multiple deploys per day	Between once per week and once per month	Between once per week and once per month
Lead time for changes	Less than one hour	Between one week and one month	Between one week and one month
MTTR	Less than one hour	Less than one day	Between one day and one week
Change Failure Rate	0-15%	0-15%	31%-45%

Reference: Accelerate - Nicole Forsgren et tal

Simplify operations through automation

- ➔ Operate based on metrics
- ➔ Invest on tooling for automation, deployment and metrics collection
- ➔ Enable accessible CD through Gitops
- ➔ Look for at least 70% of infra/deploys to be uniform, leave corner cases for data and mobile
- ➔ Invest on short feature deployment cycles with product prioritisation (lead time)
- ➔ You build you run in an standard way

Invest on cost management

- ➔ Tag and control Cloud resources
- ➔ Conduct monthly resource distribution reviews
- ➔ Adopt at least on cloud cost management tool besides the one your vendor provides
- ➔ Prepare for yearly reservation (or the equivalent) cycle and monitor its usage
- ➔ Only go for containers with a stable stack and after nailing CI/CD.
- ➔ Monitor egress and ingress traffic, cross region and hidden Cloud costs (snapshots, images, unused load balancers and gateways)
- ➔ Discipline on reservation vs spot vs saving plans

Invest on visibility

- ➔ You build, you run: drive incident management processes, automation to protect delivery
- ➔ Invest on post mortems and learning events
- ➔ Get involved on incidents
- ➔ Metric and measurement efforts to all teams

Infosec - humans

- ➔ Hire people that like infosec, curious engineers that are willing to learn. Hire experienced security engineers to teach them and your team. Software engineers learn fast.
- ➔ Look where no one is looking. Make it easy for engineers to secure their apps and services as they do with testing. Trust no one, specially yourself and webview mobile apps.
- ➔ Avoid at all costs an Infosec org that work as a barrier for people doing what they need to do. Train them to help you.

Infosec - tech

- ➔ Go for automation- it is easy to find good pentesting and code review tools that plug on git
- ➔ Look for compliance advice that reflect on engineering decisions early: PII (private identifiable data) storage can change depending on what you do and collects.
- ➔ Get interested on reading security incidents post mortems. You can find them everywhere, look for your organisation story and build them if they don't exist.
- ➔ Start with two metrics: **Incidents per month** and **time to fix vulnerabilities found by pen testing**.

Data

Data Engineering is not devops

Managing distributed databases, migrating data, draining and running queues may look as an infrastructure problem but it requires a different set of tools and knowledge.

"80% of a data scientist work is data engineering (data preparation and tooling) - Fabiane Nardon"

Create discipline on tooling to avoid big migrations (anything over 10TB is hard to migrate timely in the cloud)

What to look for

- ➔ Look for managed cloud solutions but evaluate the lock-in, employ cross-vendor managed primitives (Object storage, queues)
- ➔ Adopt a common data format (parquet, avro, json)
- ➔ GDPR-like regulations are here to stay, implement how to forget data, record the data ancestry
- ➔ Archive data by at least two transformed dimensions: e.g. date and customer id
- ➔ Work on model deployment from day one
- ➔ Push work where work happens: Data Analytics and Science within squads, infrastructure and engineering building shared platforms. Avoid big project offices.

What to avoid

- Too many of of each building block (e.g. standardise databases and caches as much as possible)
- Non partitioned databases
- Mix between events/serverless and batch processing
- Building a scheduler (use airflow)
- Building a log pipeline (use ELK, use Athena/S3)
- Teams too disconnected. Look for synergy when learning what the following teams do:
  - Data engineering
  - Analytics (BI)
  - Data Science

Closing advices

Healthy teams

- ➔ Span of control: 5 to 8 people for each manager
- ➔ Line managers with small teams, set planning rituals
- ➔ Engineering managers supporting line managers with clear metrics and quarterly planning
- ➔ Standard stacks and technology
- ➔ Management training process: "An Elegant Puzzle" can provide common ground on what is expected from a manager

Teams and culture

- Assess healthy and unhealthy teams
  - Teams with less than 3 people, lead included
  - Teams with high churn may lack purpose or leadership
- Set the management culture right, ensure a good onboard process
  - 90 days to be productive
  - First week should be directed to learn business, platform, how to deploy, finish your setup
- Hire for good teams that solve critical business needs, not for sales, features or ad-hoc projects
  - Check the culture before putting up referrals
  - Look for diversity

Be mindful

- What brought you here is not guaranteed to work in the future
- Look how the new generations work, learn with them
  - Drop the "millenials" bs
- People is important, technology is a side-effect
- Money can't buy everything but helps a lot
- Don't be petty, don't save on tools that help your team
- You are the outsider, people always think the past is better than it really was. They will probably blame you for change at some point. Breathe.