Organise your engineering teams around the work by reteaming

Abhishek Tiwari

Citation: *A. Tiwari*, ”Organise your engineering teams around the work by reteaming”, Abhishek Tiwari, 2019. doi:10.59350/e0r9t-mfq72

Published on: July 20, 2019

When it comes to organising engineering teams, a popular view has been to organise your teams based on either Spotify's agile model (i.e. squads, chapters, tribes, and guilds) or simply follow Amazon's two-pizza team model. On a positive note, both organisational models focus on having small yet productive independent cross-functional teams and there is nothing wrong to draw inspiration from either of these models when building your engineering organisation. Both models work on the scale, particularly when the company's core values, priorities, and culture all three are well aligned. Now, this is where most companies fail, either by putting too much focus on core values and culture but forgetting about business priorities or just focusing on business priorities but not enough stress on core values and culture.

Don't be trapped by dogma

A lot has been said and written about why something worked for Spotify may not apply to your organisation, and the purpose of this article is not to go in those discussions but to highlight practical issues with these models. One thing stand-out to me is need to be intentional and practical about your engineering organisation design. First and foremost, being intentional about organisation design requires good and honest discussions about all possible options. Secondly, you need flexibility and room for error when iterating and working with your organisational structure. Lastly, as an executive or leader, you should never get too attached to dogmas. This is one point I can't stress more. Wikipedia defines dogma as a principle or set of principles laid down by an authority as incontrovertibly true. And there lies the problem. As Steve Jobs wisely said,

> Don't Be Trapped by Dogma – Which is Living With the Results of Other People's Thinking

In my view, technology executives and engineering leaders are overly obsessed with the Spotify model. I strongly think that executives and leaders need to be very conscious of their choices. Instead of blindly following what everyone else does, they should make decisions based on priorities and practicalities.

Over specialisation

One side effect of not being intentional about the organisation design is specialisation starts becoming teams of their own. Specialisation creates silos which mean teams' learn and evolve within their own silos without enough cross-pollination of ideas. Specialisation could be around products, business process, or technologies. Let's take an example of retail as a domain of interest. One way to create a Spotify model inspired engineering organisation is to organise long-lived squads by retail business process hubs - i.e. specialisation around business process. As a bare minimum, I can think of an engineering organisation of 6 Spotify like squads with each team consisting of 8-10 people including

engineers (frontend/backend), BA, PO, and an agile coach. That's an overall engineering organisation of 55+ people including tribe leaders, chapter leads, etc.

- Warehouse engineering squad - managing software services related inventory, stocktake, dispatch, allocation, transfer, robotics, etc.
- Customer experience engineering squad - focus on end-to-end customer life-cycle, marketing, targeting, personalisation, loyalty, etc.
- Store engineering squad - focus on software and systems required for the storefront including point-of-sales system, promotions, etc.
- ERP engineering squad - supply chain planning, purchase order management, product lifecycle management, merchandise planning, etc.
- Back-office engineering squad - customer support, business intelligence, real-estate management, systems for finance & HR, etc.
- Supplier engineering squad - focus on supplier marketplace, demand and replenishment forecasting, risk and compliance, etc.

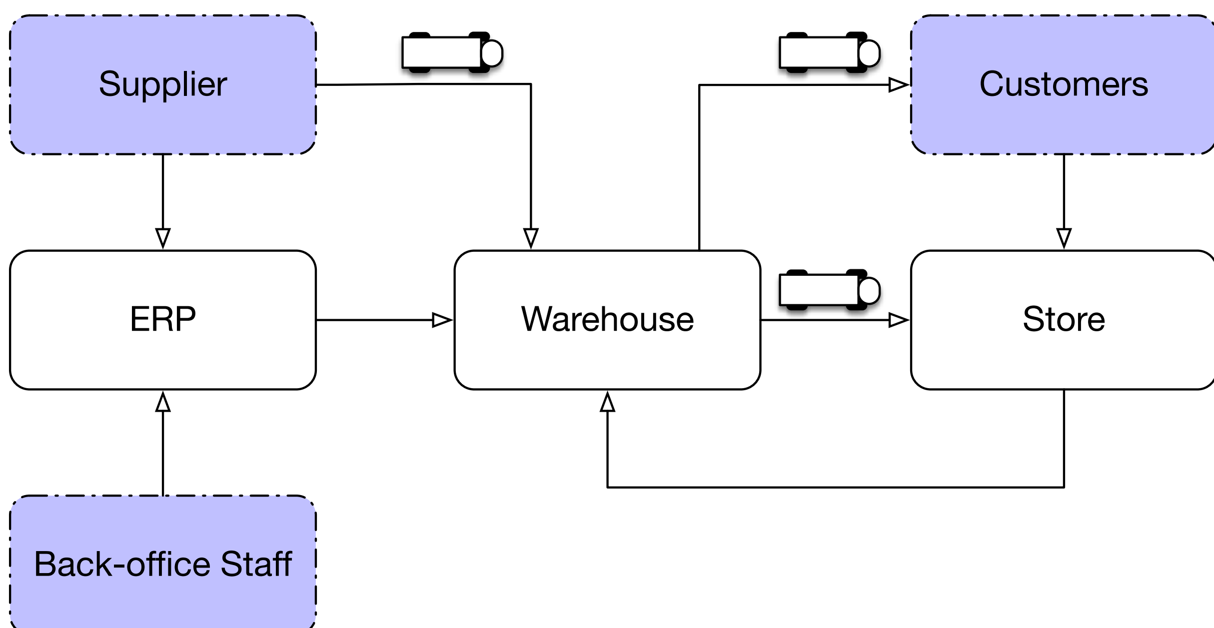

Figure 1: Retails domain - creating engineering organisation around business process hubs

Now one may ask - is this the most effective engineering organisation for a retail business? Certainly not. It is one of the ways you can organise your engineering teams in a retail environment. The right answer mostly depends on the triad I mentioned earlier - core values, business priorities, and culture. For instance, if you are fast-growing VC funded e-commerce startup and your number one business priority is multiplying current growth and performing exceptionally well on key financial metrics charted

out by your investors. In the current environment where business ideas can be easily cloned in months and features can be copied in days, you want to move fast. The engineering organisation described may not work for you because of a team of 8-10 people is still a very big overhead.

In this model, software architecture and code ownership is a reflection of the organisational design. So when it comes time to doing work which has deep dependencies generally you have to figure out which team is going to do what. This leads to endless meetings where engineering management get involved to discuss what's to be built, how to break up dependencies in manageable chunks and delegate them to various teams.

Over specialisation is considered good in industries such as healthcare and aviation but in software engineering over specialisation can be a blocker. Unlike healthcare and aviation where practices don't change over the decades, software engineering and related technologies are changing every day.

Long-lived procrastination

Most of the software engineering organisations are designed around long-lived or stable teams where teams composition and focus (i.e. product) don't change over a long period. The very idea of long-lived teams came from Scrum (see LeSS and SAFe agile guides). Both Spotify model as well as Amazon's two-pizza team model don't say anything explicitly about longevity of the teams composition and or focus. In fact, Spotify has adopted a more fluid team model.

In my opinion, long-lived teams are generally good at many things. For instance, over time

- long-lived teams get better in terms of predictability - i.e. considering a similar type of work given repeatedly over the time you will see less variance in team's agile estimations.
- long-lived teams get better in terms of internal communication and expectations management - i.e. each teams members know what to expect and how to communicate with other members.

Sadly, long-lived teams are equally bad on several things such as,

- long-lived teams are particularly good on procrastination. Either they try to build perfect products or worse use their time to perfect their code by excessive re-factoring and re-engineering.
- long-lived teams get bored very quickly due to the repetitive and homogeneous nature of work which they channel out by spending time on things like open source projects.

Team performance model such as Tuckman's Team-Development Model (Forming, Storming, Norming and Performing) don't factor aspiration and career goals of individual team members. Tuckman's model was published in 1965, since then our workplaces have drastically changed. Although Tuckman's model is still useful to understand team dynamics, it's not an immutable law of team evolution. A case in point is hackathon events (aka Hack days or Shipit days) which typically attract diverse set

engineers - mostly stranger to each other - yet forming, storming, norming, performing all happens within 48hrs. How is that even possible?

Is it possible to draw inspiration from outside of software engineering? Probably yes. In industries like aviation, it's impossible to create long-lived cockpit crew composition. Airline crew pairing and composition problem is an area of deep research. Crew composition and pairing need to work with aircraft schedule as well as crew's punctuality. For instance, if one of the pilots get sick, the airline still has to operate flight by allowing a replacement pilot to take over. Luckily, aircraft operating manuals and training procedures are so formalised and well established that there is no scope of performance degradation even if one or more crew members are replaced. Now, one can argue that unlike software flying aircraft is a repetitive task. But again, when flying aircraft, stakes are way too high to the point that a slight miscommunication or misalignment in crew expectations can be fatal.

I also have a strong feeling that long-lived teams are not good for innovation and disruption. As time passes, teams get in their comfort zone and can't think beyond their cocoon which generally comes with over specialisation and silos.

Contrarian view

What I am proposing here is a set of principals to change how you deploy your engineers to do their best work in a truly fast-paced environment. Each of these goes against the popular opinion hence why they are worth your time.

- Organise your engineering teams around the work by reteaming
- Optimise your engineering organisation for velocity over predictability
- Avoid the illusion of progress - being very much directional all the times
- Create a culture where everyone is directly involved in value creation
- Select a technology architecture which scales with your organisation
- Have cooling-off mechanisms in place to avoid burnout of your best talent

Organising around the work

Organising your engineering organisation and teams around the work by reteaming in short-lived teams and not in long-lived streams. Design your teams to stay small and iterate fast. Start by creating delivery teams of 4-5 people, basically cutting the size of traditional Spotify squads to half. By doing this you are cutting communication overhead and process problems roughly by 4 times (assuming communication overhead and process problems are proportional to [n * (n - 1) / 2], where n is group size). Below are some of the rule on engagements when organising around the work,

- Firstly, keep the composition of these delivery teams of fluid and flexible to the point that teams are short-lived (as a rule of thumb 6 months at max). Because you are changing team composition, you need robust norms of conduct and engineering practices in place.
- Secondly, fine-tune team composition based on work. Depending on work you can choose a smaller team of similar expertise (for example a team with mostly frontend engineers) or a smaller team of diverse expertise (team with balanced frontend, backend, data engineers).
- Thirdly, let engineers themselves choose the delivery teams and organise them around the initiative. Teach your engineers how to do teaming, reteaming, and onboard new team members. Encourage high-collaboration practices by running regular hack days or mob programming events.
- Fourthly, favour generalist over specialist i.e hires more full-stack engineers. Encourage specialist to diversify their skills as long as it aligns with their career goals and job expectations.
- Lastly, make upcoming work or initiative visible and transparent to everyone so people are aware of what's next in the pipeline. I suggest maintaining a high-level work and reteaming wall where engineers can pitch themselves and other members for the upcoming work (see following illustration).

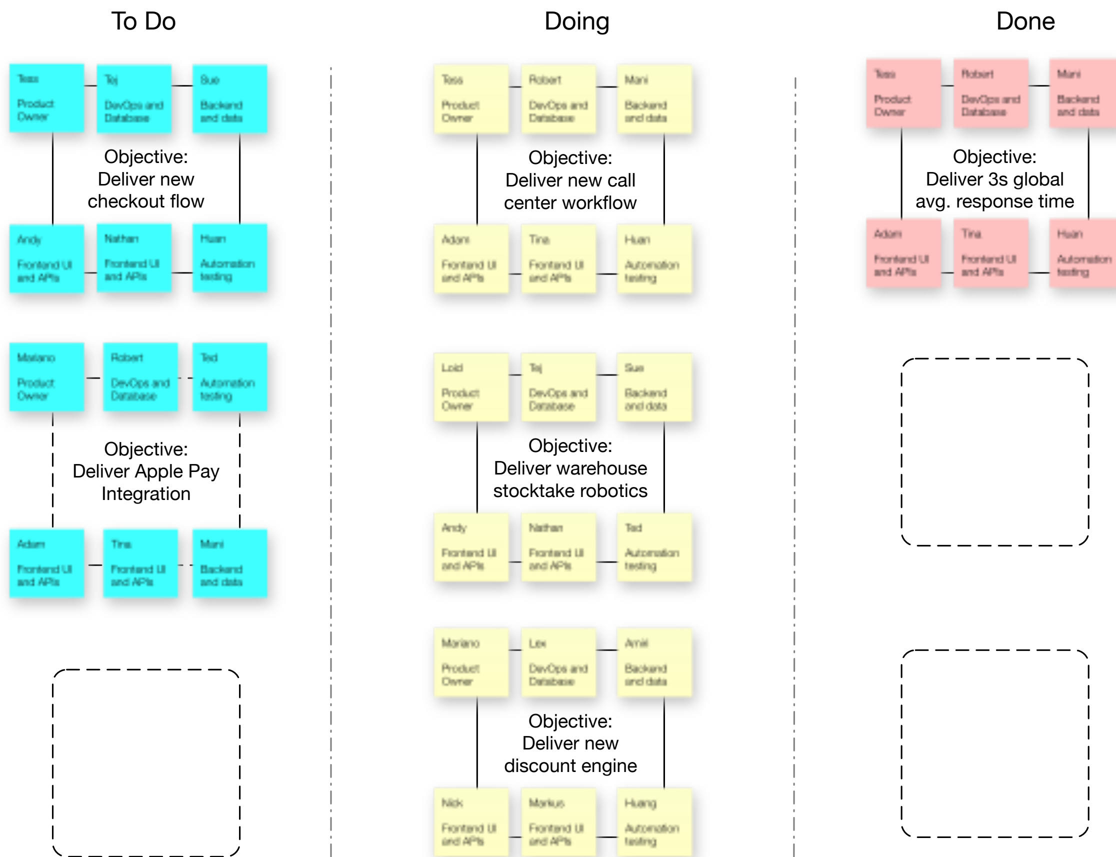

Figure 2: Organising your engineering teams around the work using fluid and flexible reteaming. A high-level work and reteaming wall where engineers can pitch themselves and other members to self-organise for the upcoming work.

Reteaming

In my opinion, reteaming allows you to change your team composition and their focus continually. Sometimes new teams are created by sourcing team members from other existing teams. On other occasions, you bring new engineering hires and jell them some existing senior engineers in your organisation. Teams created by reteaming are generally short-lived or temporary and many ways opposite of long-lived teams. Reteaming has been specially useful to support requirements of fast-growing companies, companies undergoing massive transformation, or companies trying to develop new products on an unprecedented pace.

Several reteaming models have been developed to support teams' fluidity. For instance, isolation,

one by one, grow and split (like cell mitosis), merging, switching, etc. Valve's approach to reteaming is probably one of the best out there. Valve introduced cabals, self-selecting and self-organising teams largely temporary.

Cabals are really just multidisciplinary project teams. We've self-organized into these largely temporary groups since the early days of Valve. They exist to get a product or large feature shipped. Like any other group or effort at the company, they form organically. People decide to join the group based on their own belief that the group's work is important enough for them to work on.

As mentioned before hackathon events such as Hack days and Shipit days are another great way to create reteaming culture in your engineering organisation. In my own work, I also had very good success with technology boot camps particularly useful for one by one reteaming. We ran technology boot camps every quarter, aligned with our quarterly goal (OKRs) cycles. We used tactics like war rooms i.e. isolation reteaming to deliver urgent product initiatives. Similarly, many companies regularly do mob programming to promote reteaming.

Velocity over predictability

I covered this briefly in one of my recent blog posts (see [1]). Agile teams optimised for velocity generally deliver more value per release. Why this is so important? Building the right product is as important as delivering the product on time. Technology is changing at a very fast pace which makes most of the product companies vulnerable to disruption and getting cloned. SaaS products can be easily cloned in months and product features can be copied in days. If you aren't the fastest company you essential start losing market share. Just look how Microsoft Team - which initially started as a Slack clone - is now surpassed Slack in terms of daily active users. Microsoft Team now reports 13 million daily active users more than 10 million people who use Slack daily.

Avoid illusion of progress

If you're going in the wrong direction, it doesn't matter how fast you're moving. It's important to understand the subtle difference between speed and velocity. Velocity is directional (a vector in mathematical term). Similarly, distance covered by directionless speed is just an illusion of progress. Real progress (or displacement) comes when velocity is applied in the right direction. That's when you see things get done.

If only agile metrics were vector but unfortunately they are not. Agile metrics are scalar values with zero or minimal indicators on the direction which is why they give the illusion of progress but real progress hard to recognise.

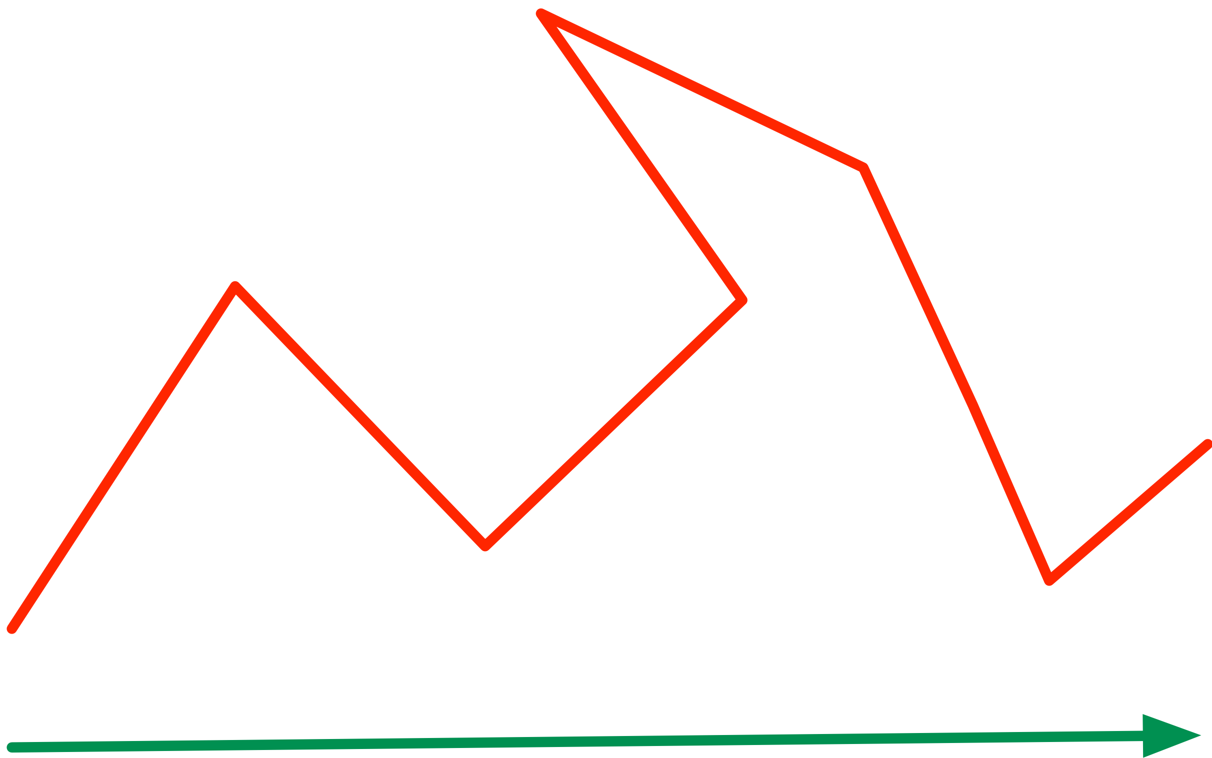

Figure 3: Direction over speed and why you should avoid illusion of progress

De-coupled and stateless architecture

Select a technology architecture which scales with your organisation i.e. an architecture which enables the scaling of your engineering organisation easier. Just like architecture, teams should be loosely coupled or better completely de-coupled. This requires well-documented coding standards and architectural design patterns so that standards and patterns are well aligned.

Just like architecture teams should be stateless i.e. less cognitive overload of context so they can move fast. This happens when everyone in your organisation understands your products and offering well. Remember code ownership is overrated. No one remembers the code they wrote 3 months ago forget

about 6 months or a year. What they do remember is the problem they solved and the approach they undertook to deliver the feature.

If implemented correctly, microservices pattern sufficiently decouples engineering teams and mostly parallelise their ability to build new things with less blockage and dependencies. In similar vein patterns like single spa, micro-frontend, shared UI components as npm packages, can parallelise the frontend development work. With the emergence of versioned serverless functions and per-branch environments, teams have several tools on their disposal to unblock themselves.

Everyone is a value creator

You lean down your leadership and management layers, and set a clear expectation that everyone should create value by directly being involved in actual work (i.e. doing one of the below),

- contributing by writing software
- performing code reviews
- driving technical architecture
- pairing with junior engineers
- helping to test software
- involved in writing stories
- designing the UI and UX
- preparing product launch
- creating work backlog

In this new landscape, your managers and leaders have to find ways to add value - activities such as delegation, co-ordination, etc only create marginal direct value.

Cooling-off mechanisms

Probably one of the biggest drawbacks of organising your engineering teams around the work is burnout of your top performers, particularly your rockstar engineers who are continuously looking for challenging works. For some top performers quarterly goals (OKRs) can be addictive. You need to have preventive measures to avoid burnouts. One of the options is to rotate your engineers and managers on a 3 months long production support stream where they are rested from any aggressive goals and timelines. Resting in sports is a very common practice. This allows to recover or recharge individual for optimal performance.

Closing thoughts

I may have not tackled some of the burning questions particularly how to support or maintain software when teams composition is so fluid and dynamic. Some people have suggested that long-lived teams are better on maintaining software which is never backed by any data. Again code ownership is overrated so we may be overestimating the benefits of long-lived teams concerning maintenance. Some companies took all hands on deck approach especially around major issues. Others use a production support stream. There may be more creative ways to perform maintenance activities but I let you do your own homework.

{{< youtube 9UyRxFDe0yI >}}

References

[1] A. Tiwari, "Agile: Optimise for Velocity Over Predictability," 2019, *Abhishek Tiwari*. doi: 10.59350/m6kre-nkk76.