KOMMITMENT

Product Development Flow - how to scale beyond 20 developers

&

Verticalization

Author: Hannes Mainusch
Date: 2019-11-08, 2019-11-18
Version: 0.42 - medium rare, not well done yet
Edited by: Reshma

tl;dr

Scaling product-development organisations beyond 20 people of roughly beyond the size of two development teams requires an understanding of how organisations work. And in the field of IT, it requires specific experience in dealing with stuff that is hard to communicate to non-nerd communities. The two principles that make IT-product development more difficult to organise in comparison to typical manufacturing or service organisations are:

1. The intangibility of the product. IT people produce lines of code and lines of code are not readable or communicatible to most non-nerd people.

2. The speed of IT evolution. IT changes so fast that it is seldom that we repeat the same thing frequently.

Due to these two things it is hard to plan and optimize. And it is also hard to get an overview of what everybody does in an organisation in detail. The patterns described in this article help to become fast and adaptable company that has the fun and agility of a startup and the level of organization and efficiency of an enterprise. At least, that's what we're aiming to do...

"Verticalization" is a term used in the organisation of product-development teams that has evolved over the past years and addresses a change in the way we understand organisational and architectural structures. In contrast to more classical organisational layouts of teams being responsible for individual domains of technological specialisation like the traditional "operations", "DB & persistence", "Network", "Security", "Backend-development", "Frontend-Development", "Specification", "QA" teams in classical IUT-organisations. Vertical teams combine many of these domains in one team and orient themselves in the vertical business areas, for example "user identification", "shopping basket", "product", "Sales", "Aftersales" in e-Commerce. Other domains, like Retail or Banking will have other vertical teams due to their different business models.

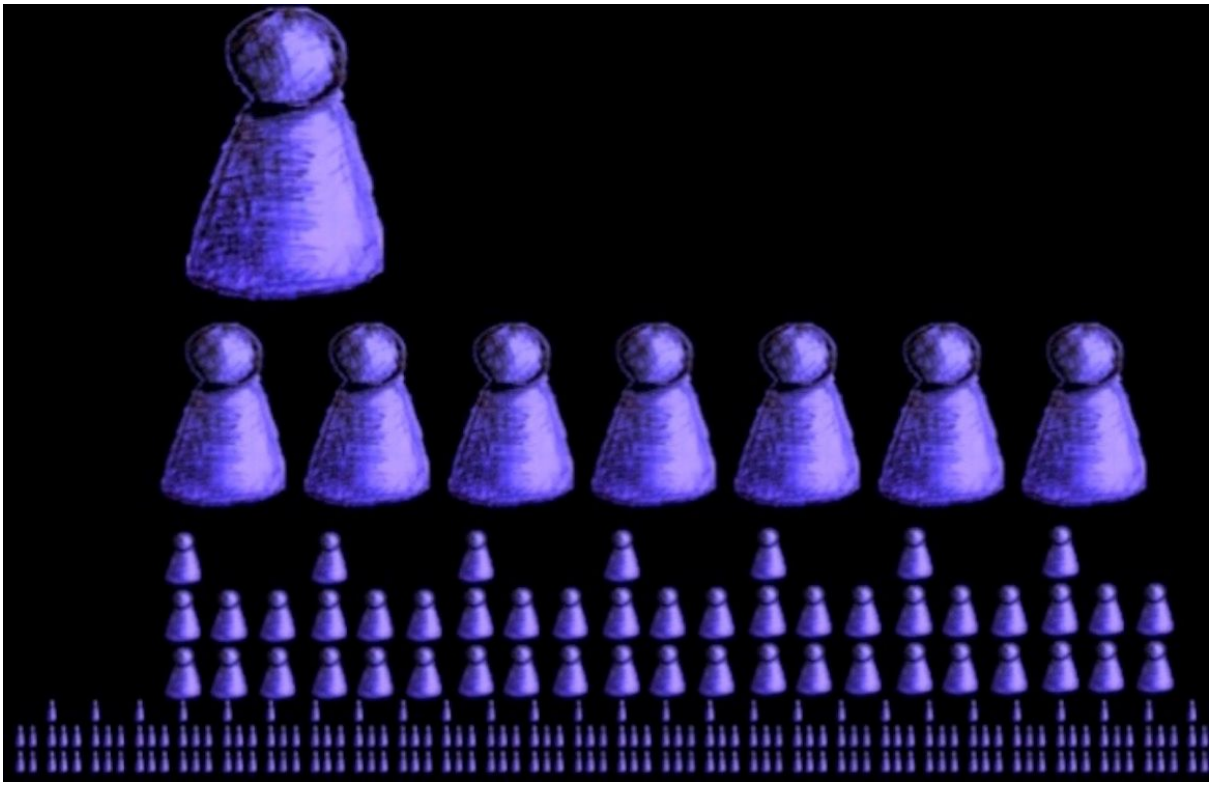
Verticalistion has proven in various enterprises to lead to independent and autonomous teams with high productivity. But whatever effort is made to disentangle teams, there will always be residual team-cross-cutting concerns, like shared libraries or epics that touch multiple teams.
The remaining cross-cutting concerns in vertical organisations are dealt with in committees, tribes, COPs or InnerSource methods and patterns. Temporary cross cutting concerns are dealt with by using Portfolio-Kanban rather than the mulit-project-management approaches that were popular in the 1990s.

Despite being a successful and proven concept, verticalization, portfolio-kanban and the role of management in all of this are not yet commonly understood or accepted organisational patterns. Thus this article.

Why should we describe our future organisation?

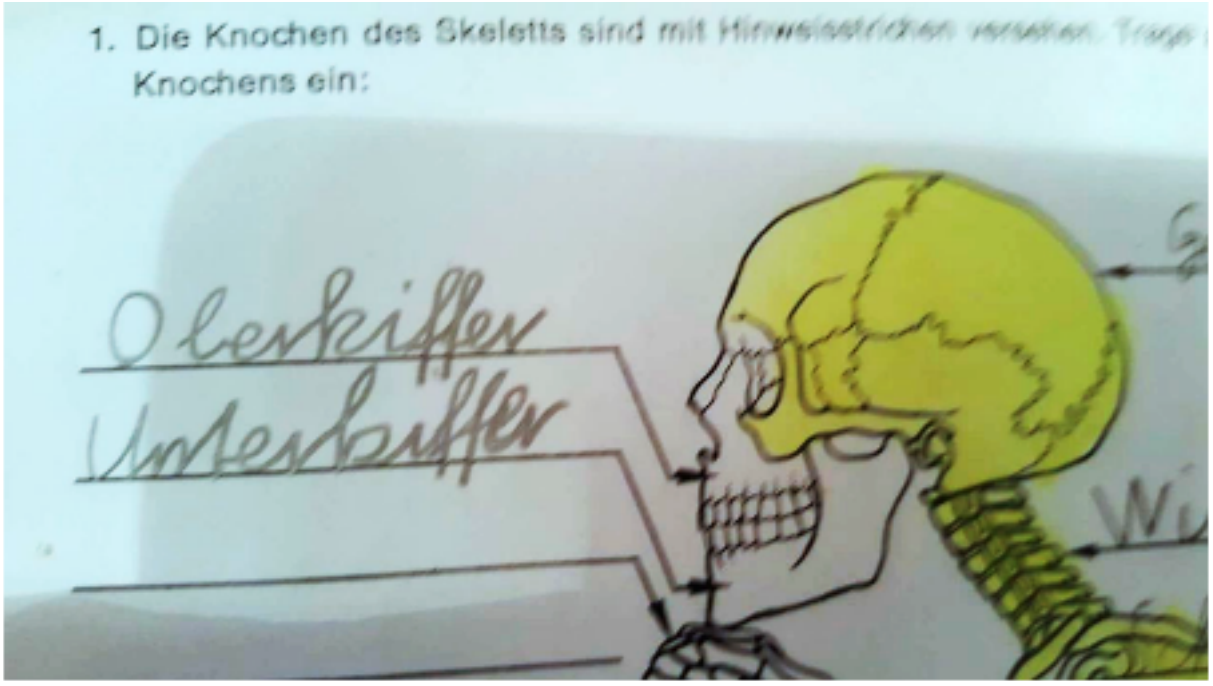
If we do not describe what we aim for explicitly, we do not give our new employees the chance to understand what we aim for. We would not have status quo that could be improved by arguing the cause. We would still rely on the bonfire, like an oral transmission of knowledge in team meetings of 10+ people, in the hope that all these “bonfires”, stories are being told in a traditional manner with the same aim. We would still rely on a one boss communication from top to bottom of 100+ developers.

Luckily, this domain is over and we are at a time in history where things are documented in papers, podcasts, videos and wikis. Only documentation settles differences by agreement and gives new hires a chance to understand where we come from and why we are where we are now. With this understanding comes great responsibility for solid change.



I think organisations without documentation culture are still in the stone age of prehistoric development. When organisations with this culture deliver software, they die the death of legacy code, workforce migration and language changes. Often enough, users later uncover the lost knowledge with the fine and careful brushes of archaeologists in order to not destroy these fragile

structures of legacy in operation while trying to recover knowledge without destroying the system.
I debate the second Agile Manifesto of "working software over comprehensive documentation". I believe, comprehensive documentation is a prerequisite of working organisations.



What is the problem?

In traditional organisations we observed that the past decade of IT and product-development have led to situations where companies have created huge central IT-Systems with millions of old and unattained source code which becomes increasingly hard to change and maintain. At the same time the IT workforce ages and the risk for companies to maintain their business increases. The following list demonstrates a couple of examples of software legacy in Germany, which have crossed the authors career in the past year:

- **Automotive**: 15+ years of Java based legacy code in vital container logistics system. Source code was deleted many years ago and documentation is sparse.
- **Banking**: Central Cobol and DB2 based core banking application.
- **Insurance**: 30+ year old systems running on Siemens BS2000 operating system. Contracts on this system will still have to live for another 30+ years.
- **Container Logistics**: 20+ years of Cobol based systems which have not been fully migrated to the "new" J2EE based system which was developed in the past 10 (!) years.
- **Toll collect systems**: 6 months of change iterations on 15+ years of Java systems.

These are the inspiring success stories. The various unsuccessful attempts which have been discussed in the media are not mentioned.

At the same time, the average age of the workforce in the industry is above 42 years of age with the Baby Boomers eagerly awaiting their coming pensions.

In most companies, this is a hidden risk to shareholders and the board of directors which is not covered by regulatory means. The annual reports of most companies do not contain software achievements as assets, although a simple arithmetic and comparison change within companies that publish their digital assets achievement (e.g. XING, zalando, ...) suggest that each line of source code should be attributed with a value between 15 -35 €[1].

The methodology of change of software as an achievement has been a waterfall approach from the organisation. A pattern that has been proven very successful in industrialisation. Organisations optimize the critical path of trivial and easy to understand products, but in a complex and hard to communicate environment of intangible IT-development, simple line organisations, simple divide and conquer strategies and classical project management approaches with waterfall methodologies have not only proven to create utter failures but left us with a legacy of critical business risks.

---

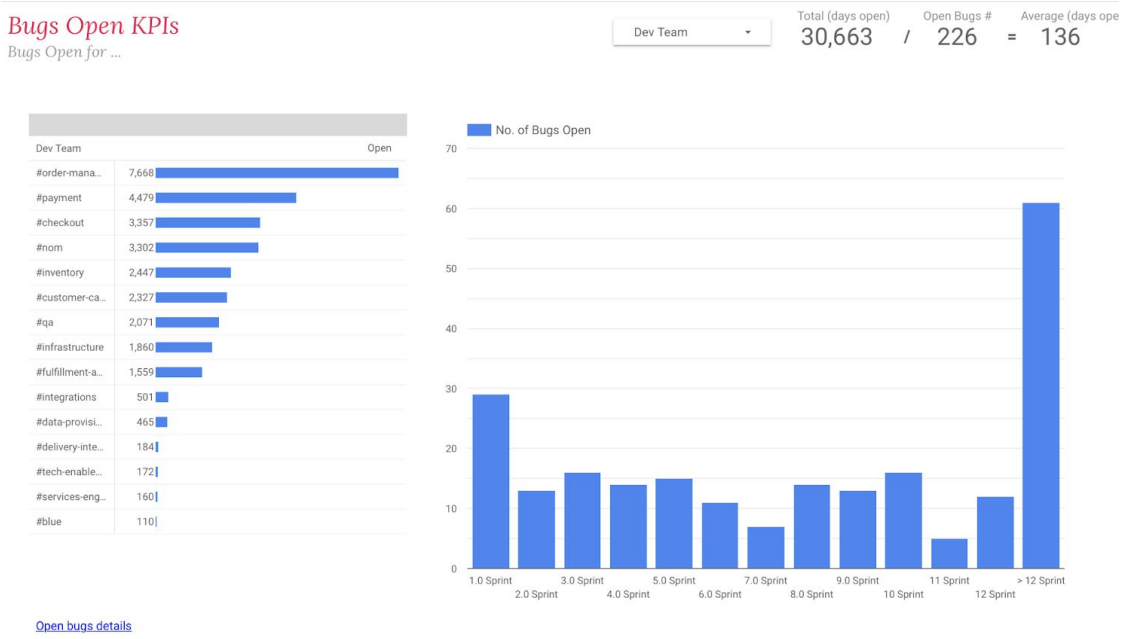
1 https://www.oop-konferenz.de/oop2018/programm/konferenzprogramm/sprecher-detail/johannes-mainusch.html

All these things have been discussed in great lengths. Over the past years many methodologies have emerged, like the agile movement, and yet organisational solutions are still hard to come by, and emerging success patterns in some business domains (spotify, OTTO, sipgate, ...) are hard to communicate and transfer to large and traditional enterprises. Even startups struggle to avoid the traps of waterfall and line organisation, especially when traditional management enters the business, the overburdened startup managers fail to evolve their expertise.

Symptoms of stuckness

Symptoms of broken product-development organisations are:

- High number of unresolved bugs
- Long running and flaky tests
- Long backlogs of old and unstarted stories
- Painful and infrequent deployments
- Production outages due to changes on the platform
- High turnover of staff and long standing open positions
- Command and control is valued over learning and gaining expertise



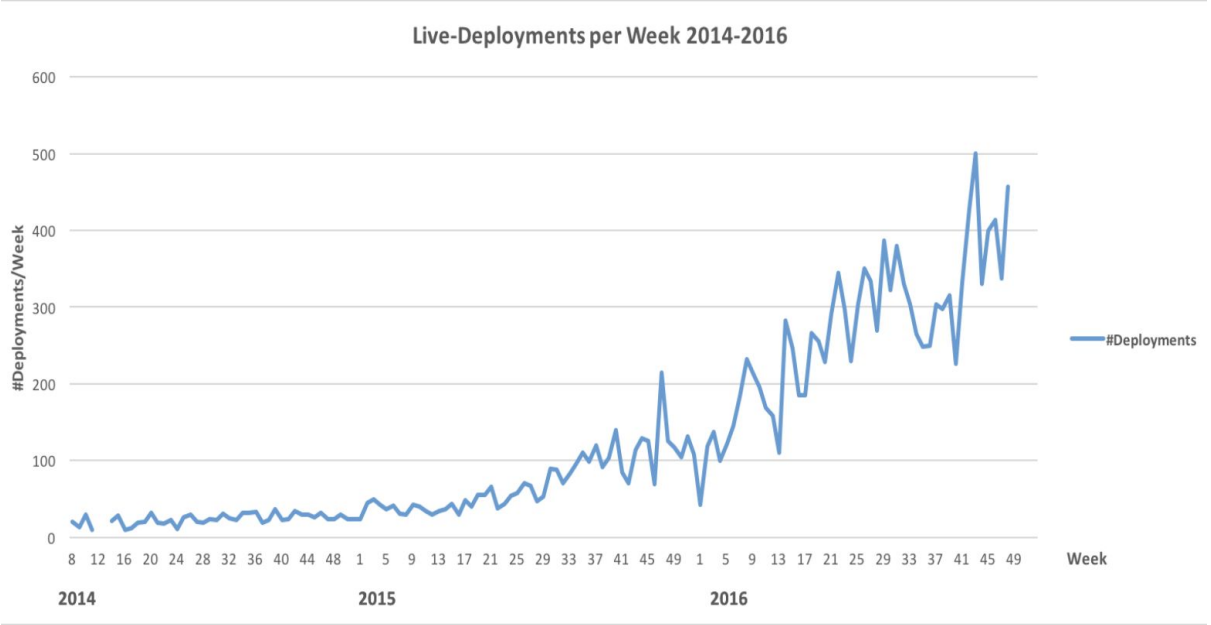
Symptoms of health

Symptoms of a healthy product-development organisations are:

- **Zero-Bugs**: A clear understanding within all teams that bugs have to be responded to and solved within two days

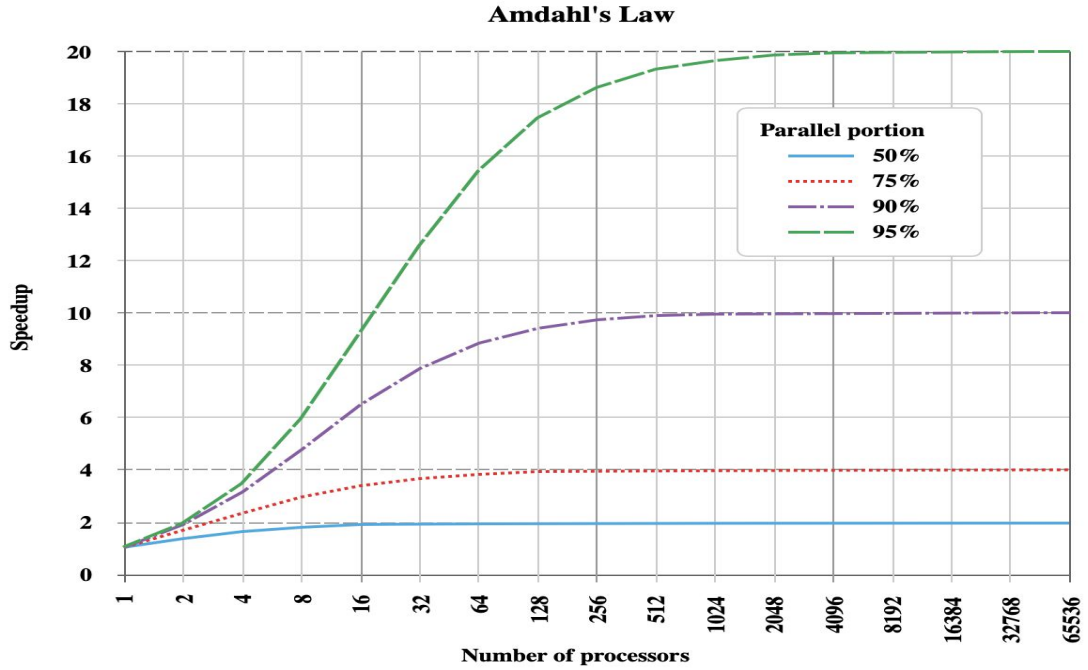
- **Portfolio-Kanban**: A solid and transparent process for working on cross-cutting concerns
- **Pull-principle**: Teams pull stories and epics, they are not pushed into them
- **Operations-Monitoring**: All teams run monitors of their products in production
- **Understanding team's purpose**: The purpose of each team is easily understood by each person in the organisation.
- **Evaluation of Epics/Stories after development**. The organisation understands the value of each produced epic and evaluates the gain of it's anticipated worth after production of it.

So the problem is to find an organisational solution out of this chaos and to turn stuckness into organisational health.



What is the business aim?

The first hint of how to organise better can be taken from IT itself, namely by understanding Amdahl's law of optimizing a parallel processor[2].



https://en.wikipedia.org/wiki/Amdahl%27s_law

Amdahl's law states that the most can be gained from parallel operating systems if the programs that run on these systems are truly independent of each other. Applying this to organisations implies a simple truth: the setup of teams and the handout of tasks to these teams should result in independency of one team's work from other team's work. If teams depend on each other, the assembly of work would require an undisturbed production line with repeatable and known handover points between units. Due to the complex nature of IT and the required source-code reading skills which only developers have, this proves to be largely impossible. In other words, the efficiency of assembly line production can only be achieved when every organiser of work is capable of understanding every step in the process. This proves to be impossible even with only IT experts in the game (think about Java/J2EE programmers talking to react native developers while having functional programming experts experts in the room). Only very senior developers with limited domains, like in large open source projects are capable of this. But with junior and non IT staff on board, this becomes increasingly impossible.

So, if not assembly line or waterfall, what then? To get an idea of modern IT-organisations, I

[2] https://en.wikipedia.org/wiki/Amdahl%27s_law

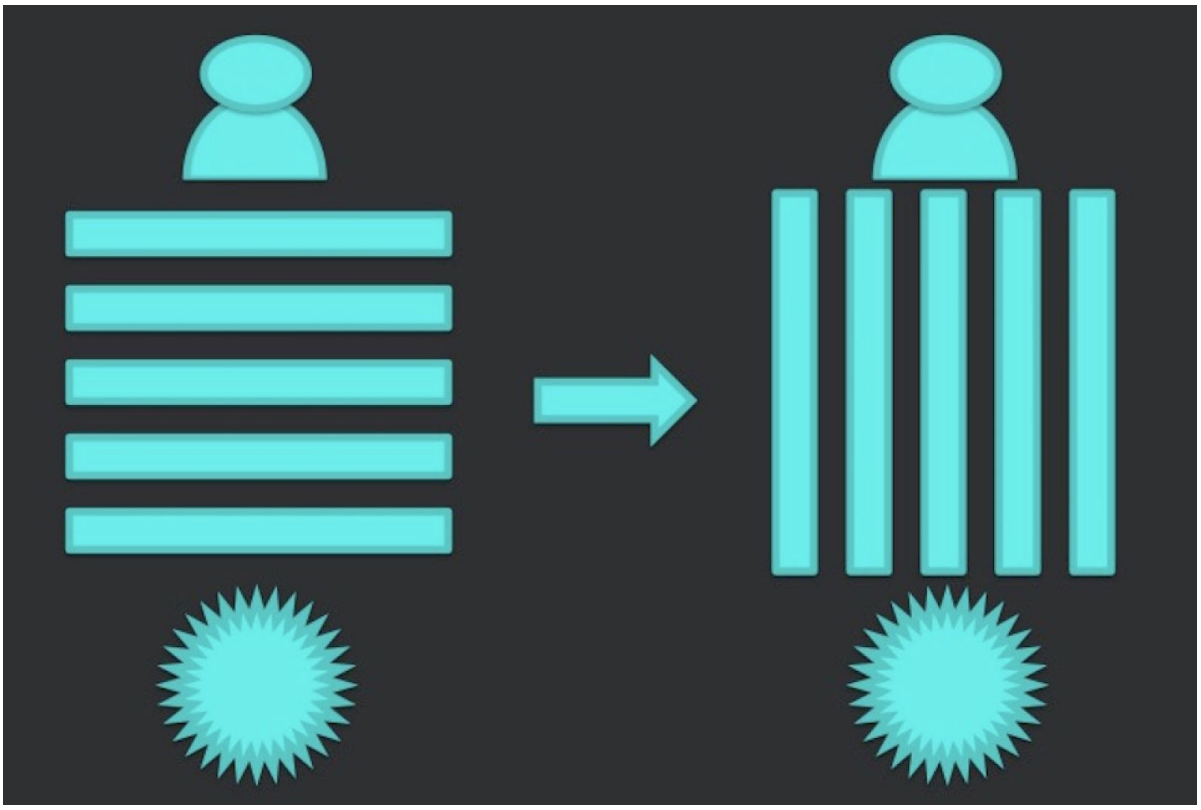
recommend watching a 5 min video by John Kotter on modern organisations combining startup and cooperate know-how[3].

While we value the efficiency of line organisations, we strive for better ways to regain effective momentum in product-development.

So the primary business aim of verticalisation is to organise teams, communications, meetings and initiatives in a way that leads to high autonomous teams, while organizing residual dependencies in a transparent way to allow the organisation to optimize. It's not so much the independent teams that slow down an organisation, but rather the interdependencies between teams.

[3] https://www.youtube.com/watch?v=Pc7EVXnF2aI

Vertical patterns



Fixing the language

Organisations and people are communication by language. To obvious to talk about? Well, no. If you ask people what the purpose of the company they work for is, they mostly have ahard time to explain it in a few concise sentences. In Germany we all love the childrens TV series “Sendung mit der Maus”, it is famous for it short movies that explain how the beans got in the can of how a Lufthansa plane flies from Frankfurt to New York. I always thought how lucky the companies are which have tis TV team on site producing next week’s Sunday Maus special, that will explain how the tea got in the bags and the teabags in the package. I am sure any business with such simple helicopter view explanations of why they exist and what they stand for and how they make a living will be well suited to survive the next decade.

So why is it so difficult in product-engineering to explain what we do in simple terms? This is partly because the terms are sometimes too simple, like “order” will mean something completely different in e-Commerce, Retail and British Parliament. Just because we know the word does not mean we understand it's proper meaning in a specific business's context. Rarely have companies good comprehensive documentation on their processes - and a respective glossary. But talking is not so different from programming computers. If you misspell or misuse words, organisations get buggy and do unexpected things. The only thing is, that computers are less tolerant if you use bad language.

Fixing the [natural] language of a company involves having up-to-date process documentation, glossaries, and time for the elaboration, of what “order”, “consignation storage location” or “Tallyman” actually mean in the companies context. In contemporary companies with cross-cultural teams and intercontinental locations it also involves understanding cultural differences. The same “yes” might mean “yes” in Hannover but “no” in Mumbai. And a negotiated and closed contract might be reopened the next day in Kairo for some minor changes.

As humans, language forms our cultures and spurs the development of societies. But in the age of microsecond transactions and reinforcement learning algorithms replacing call center agents we should recognise our inferiority towards the machine and try to put a conscious effort into avoiding

misunderstandings wherever possible.

Understanding the stakeholders and business landscape

Any business can be described by which service or product it provides for which stakeholders. One of the very important things to organise and speed up companies is to have an easy to comprehend and high level description of what the business business is. The algorithm is the following:

- describe the customers (in this context customers are the ones who pay money directly to the company or have contracts with the company. Customers / Stakeholders are the people who the verticalized organisation has to answer to. Sometimes the stakeholders are just the customers, as in many e-commerce companies. Usually in B2B settings it is more complicated.
- describe the business case. The business case is the point of exchanging product for money.
- describe the product(s)
- describe the user journeys. For each customer/stakeholder/user describe the things he/she does to do business step by step. Each of these steps could be catered for later by one vertical team.

Here are a couple of examples:

Example E-Commerce:

This is a hypothetical e-Commerce company that sells on the internet, has a partner bank that helps by giving credit to people who have little reserves on their bank accounts.

Customers:

- Shoppers: People who buy products on the internet. It turned out that the e-Commerce company can do a much better job of selling to customers they already know because that way matching products to customers is easier and also payment defaults are much less likely. It's always good to know how much a customer can spend for new products...
- Partner Bank: customers contracts, where customers fail to pay for their purchases are being sold to the partner bank. That reduces the profit on these customers but also uts the payment failure risk. Also all the rate payments are handled via the partner bank.

Products:

- The product is an item from our website that can be bought. Our products have pictures, descriptions, prices and availability.
- Our customers are also products if they enter rate payment because our rate payment subsidiary bank earns money by handing out loans to them with rather high interest rates.

Business case:

With our business we can sell products to customers via the internet, get to know and distinguish easy paying from hard paying customers and sell the hard paying customers contracts to our partner bank. Our business case is to produce customers who buy our product.
Note: in this business case not the "product" we sell to customers is the thing that is being produced, but the customer. Most Internet businesses produce customers/users/shoppers/social-network-participants/advertisement-watchers but many fail to set up the organisation properly around this production line.

User journeys

- as customer I discover a product
- put the product in my shopping basket
- check-out the product
- pay the product
- in the process I might log into my account (or not, in case of anonymous checkout)
- As user I track and trace my product
- As user I return my product
- As User I create/update/delete my user account.
- as Partner bank I get a new contract
- as Partner Bank I pay for a contract
- ...

Example Banking:

tbd

Example Retail in the consumer product market

In this example, the business is a chain of drugstore markets in EMEA. The to be verticalized organisational unit is the product-development consisting of e-commerce, clienteling and supplying the instore POS and warehouse systems.

Customers

- B2C customers in the e-commerce shop
- retail stores in EMEA
- the retail company itself for deciding about new locations and regional strategies.
- the retail companies marketing and clienteling department.

Business Cases

- Sell stuff via e-commerce,
- Support the companies' growth strategy
    - with location data
    - with customer data

support local stores with instore solutions

Products

- **Beauty-Products**: a nice mobile able e-commerce platform that allows to buy beauty products and pick them up in-store. Future aims are home delivery, client-aware curated shop, endless availability of upmarked products, product ratings...
- **Users**: Customer data for marketing, customer retention,s and developing an upmarket community for the diamond customers.
- **Instore solutions**: POS, shipping, ware-handling
- **4D-Locations**: finding optimal drugstore locations, pop-up drugstore, need and pricing information for special locations like airports, train stations, festivals, pop-up stores, regional differences, seasonal differences (last years Oktoberfest & Aspin consumption

User journeys

- as B2C user I do e-Commerce (like the eight stories above)

- as retail store I receive a shipment
- as retail store I use my POS-System

- as company I want to open new stores in locations where I see a lot of e-commerce but no shop nearby
- as company I want to send vouchers/catalogues to my premium customers
- as company I want to discover new product strategies with my diamond customer base

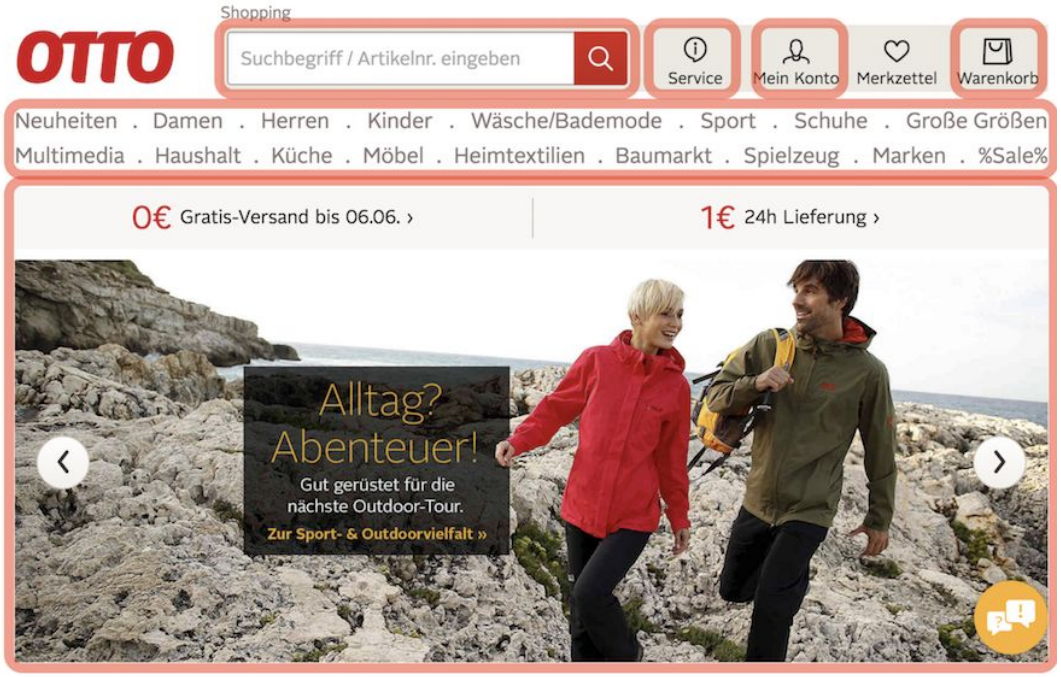
––

In reality, these cases should be better defined and agreed upon by a group of relevant product and

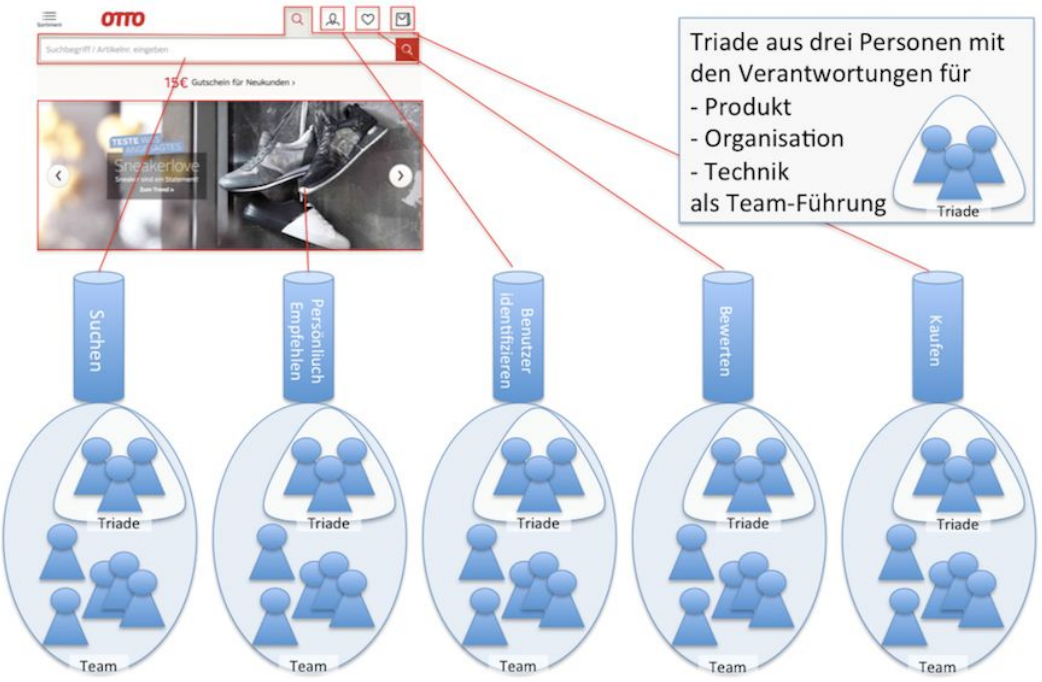
engineering staff and then brought to the attention of the whole company. This way we will get a nice simple picture of what the company is doing.

Teams to landscape mapping

This is theoretically the easy bit. Just map a team to each of the user journeys. In reality, some of the user journeys can get big. In mobile e-Commerce it is hard to split vertically frontend to backend. A vertical should ideally own all systems and processes for their piece of user story frontend to backend.



KOMMITMENT



Cross-functional teams with meaning

Team names and team manifestos communicate the purpose of the team to the outside world. Earlier, teams had names like “Research and Development”. At the formerly huge company Northern Electric, the abbreviation of this particular department name, the Northern Electric Research and Development, short N.E.R.D. That's where the infamous word **nerds** comes from.

Teams names and their purpose should be understandable to the whole organisation. So, naming teams after colors, animals, TV-series or anything else unrelated to areas of business or process can be

rendered an antipattern. Teams should have names that will allow the CFO of a company to understand and remember their purpose. This way it becomes clear to the whole organisation, to whom feature requests of bugs have to be assigned. So „payment“ is better than „magenta“ and „clienteling“ is a lot better than „A-Team“.

To even understand better who is in a team, what IT-services belong to a team, what the history of a team is and what they like on their pizza, it is recommended that teams write and publish their own team manifestos. That will also allow new team members to quickly orient themselves.

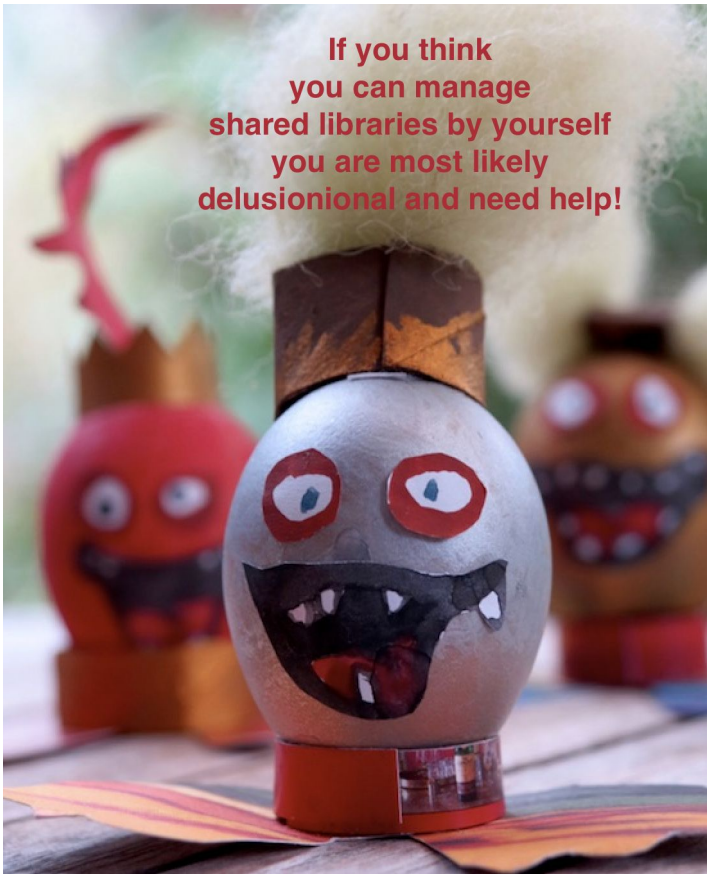
More subtle changes that organisations experience when moving to “meaningful teams” is, that when customers require new features they automatically land in the responsible teams. I have seen multiple companies that assign the next topic on the list to their next free feature team, resulting in big entangled monolithic underlying systems with no logical separation of concerns. Organisational structures with teams not bound to businessy domains usually very quickly result in monolithic IT and very sluggish processes. There, they are hard to change and fix.

The cross-functionality of teams is well known from agile structures and seldom debated today which is good. Organisations that still have their Requirements-engineering, Development, QA or DevOps in separate teams are usually slow and produce and maintain monolithic IT systems. This is for the same reasons as mentioned above. Their teams have no meaningful alignment with the business purpose of the company and no easy way to separate concerns and systems.

The shared nothing principle

The shared nothing principle is directly derived from applying the above mentioned Amdahl’s law to organisational patterns. It tries to identify and eliminate things that are shared between teams. This is because anything shared with another team invokes the risk of needing to wait for the shared resource to be free for use. Shared things can be:

- libraries, that need to be changed
- experts who work in multiple teams
- QA servers for regression tests that are being blocked by other teams tests
- meeting rooms, that are not available due to sprint changes
- DB-Servers that exist only once in an organisation…

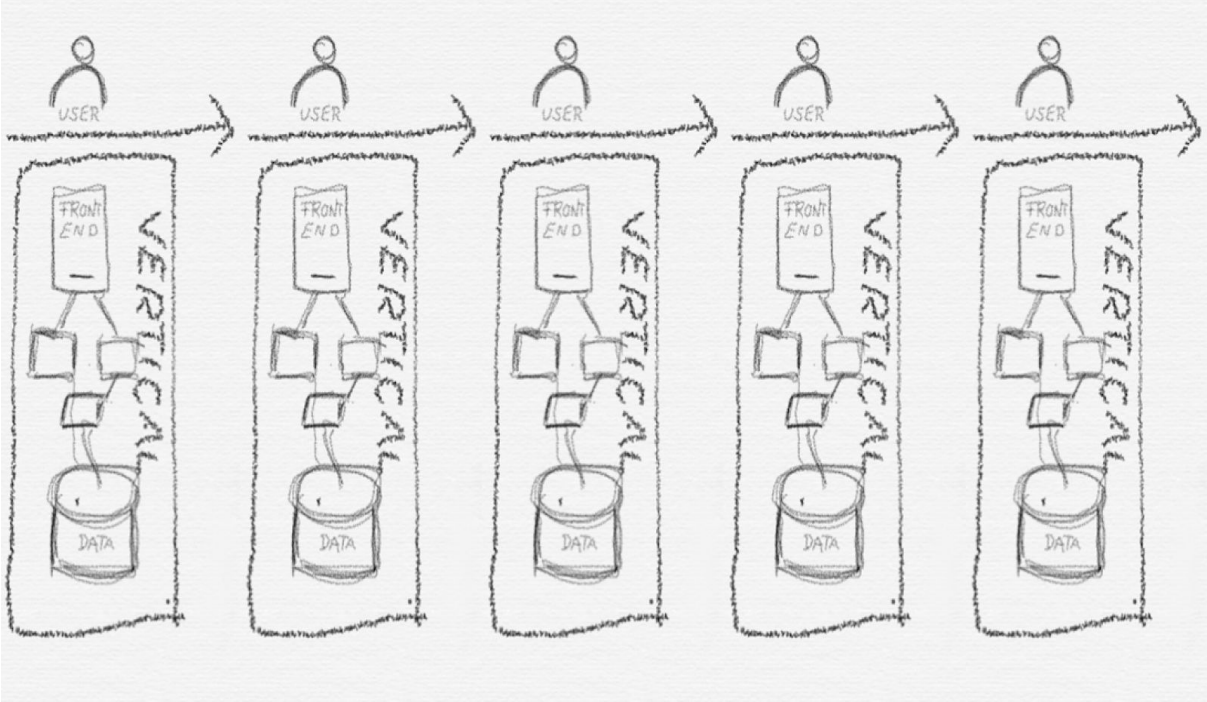


There are a lot more things that are shared in organisations. The reason for sharing are usually, because the shared things are expensive to procure for each team. In the 1990s this was true for servers and databases but today this is untrue. Sometimes architects and developers like to encapsulate complicated IT into libraries to make things "easier" or "more standardised" for general usage. But studies have shown that the average library reiuse in enterprises is well below 1.5. This means out of three libraries less than two are actually shared in practice. If you think you can do better with your libraries, you are usually mistaken. Only very few people are good in creating and maintaining shared libraries. They usually work for organisations like the Apache Foundation or other large open source bodies. So beware, creating and maintaining shared stuff should be done by at least three cooperating and very senior experts. **If you think you can do that by yourself you are most likely delusional and need help!**

Autonomous vertical team - you own it, build it, run it, fix it

This is pure devops culture. The team can only be truly autonomous if it is in charge of the whole product lifecycle. And in order for the product to be small enough to do a good job we cut the big

product in nice small vertical slices.



The handling of cross cutting concerns

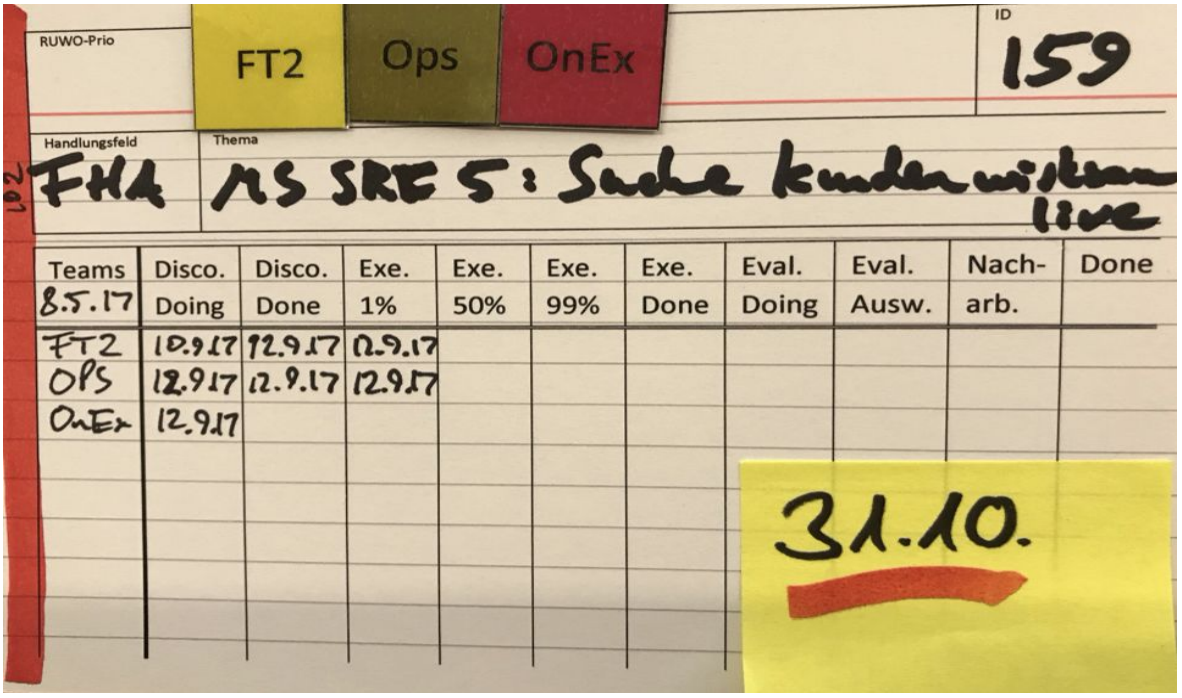
Although verticalization aims at removing the classical horizontal layers in the organization and operating product-development in teams with meaningful names and domains, there will always be residual cross-cutting concerns. Since cross-cutting concerns hinder the independence of teams they have to be dealt with with special care. There are two types of cross-cutting concerns:

1. temporary cross cutting concerns like Projects or epics that require more than one team to work on them.
2. long standing cross cutting concerns without definite end dat like the maintenance of shared libraries or Frontend-frameworks like e.g. react-native for mobile clients.

Here are best practices how to handle them differently. The general Pattern of **cross-cutting concerns have clear ownership** has to be applied.

Handling of temporary cross-cutting concerns - epics that hit multiple teams

Epics that hit multiple teams can be elegantly handled with a portfolio-kanban approach. On this kanban board every epic appears on on card, that also shows the involved teams. This way, all the involved teams can see, whether they are the one team that possibly causes the card to be stuck in the process. Temporary cross cutting concerns have to be done by all involved teams in parallel. This is the reason, why kanban is an ideal process for handling them.


RUWO-Prio
FT2
Ops
OnEx
ID
159
Handlungsfeld
FHA
Thema
NS SRE 5: Suche kunden wirksam live
Teams 8.5.17
Disco. Doing
Disco. Done
Exe. 1%
Exe. 50%
Exe. 99%
Exe. Done
Eval. Doing
Eval. Ausw.
Nach- arb.
Done
FT2 12.9.17 12.9.17 12.9.17
OPS 12.9.17 12.9.17 12.9.17
OnEx 12.9.17
31.10.

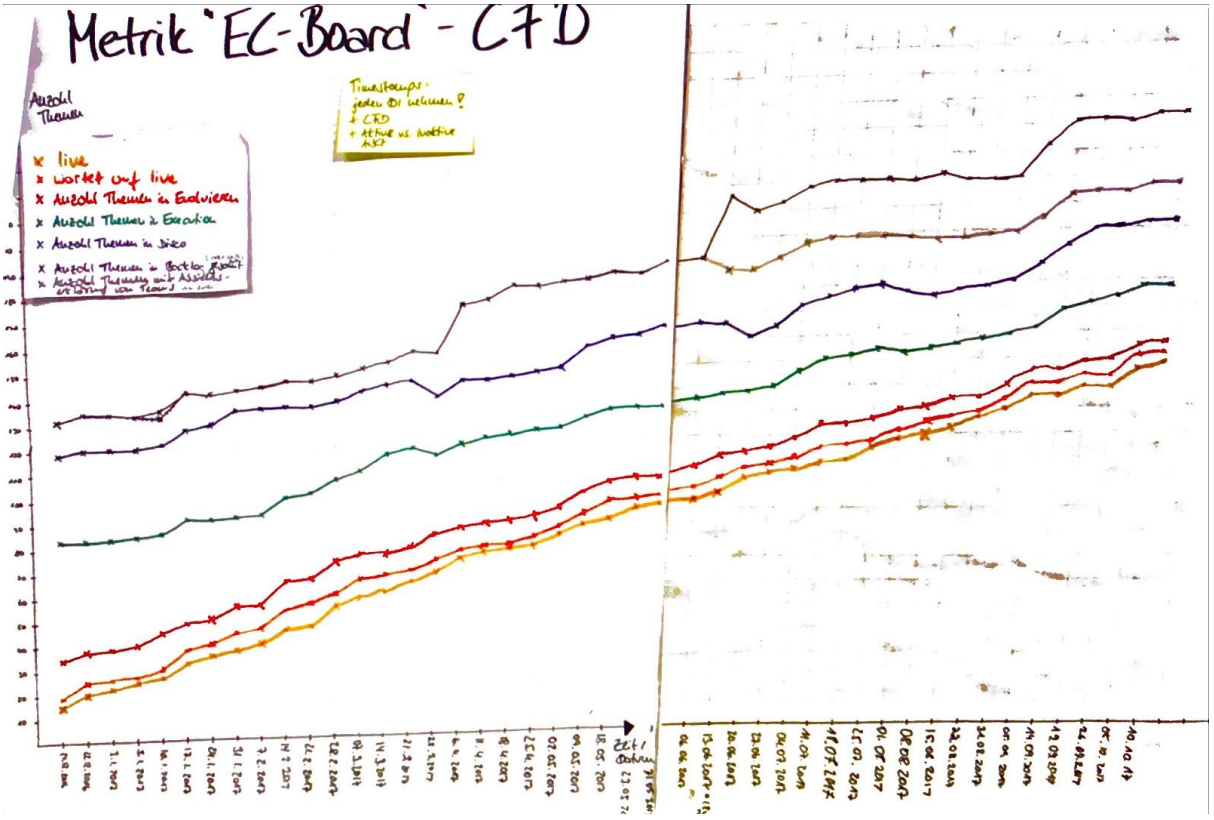

DISCOVERY
EXECUTION

Handling of permanent cross-cutting concerns - what to do with shared libraries

tl;dr
⟶ SIGs
⟶ innerSource patterns

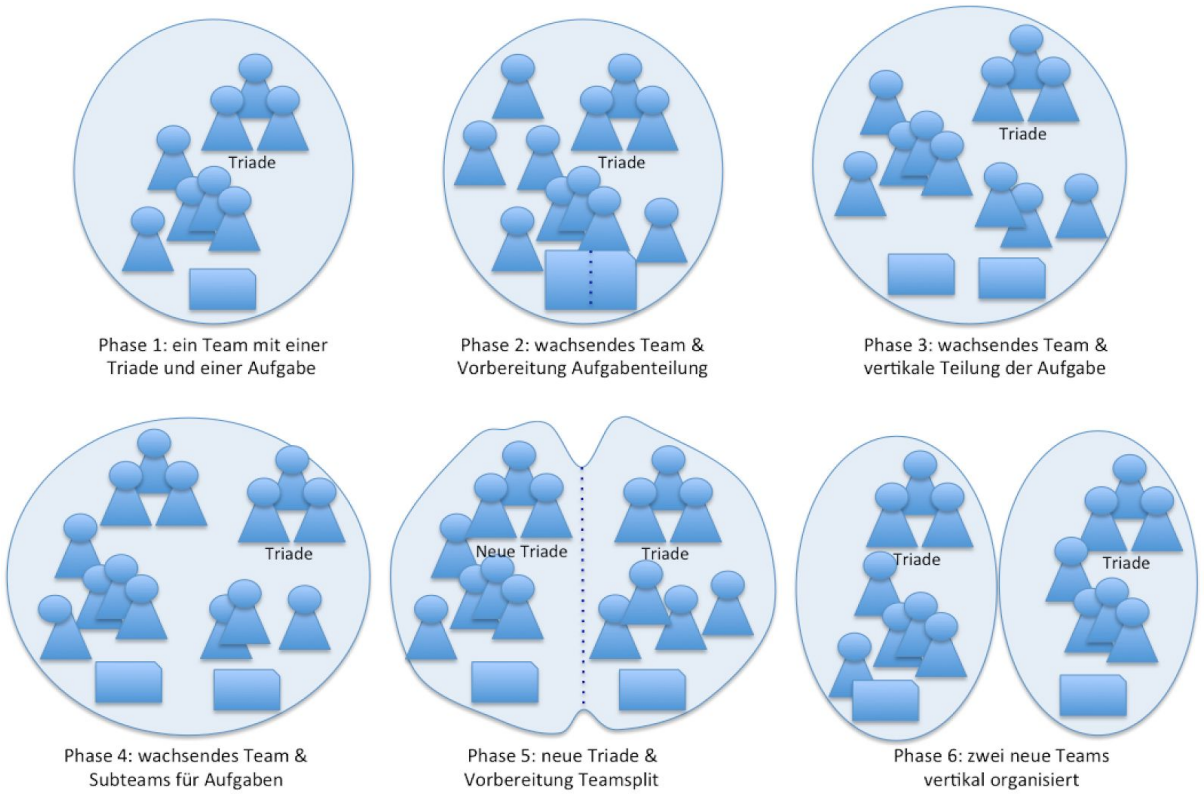
Cross cutting concerns like e.g. shared libraries should be handled (by citation of Stefan Tilkov) with the professionality and process of open source libraries. Some companies actually open sourced their shared stuff on github to fulfil this request. One working pattern is to pull people from all affected teams into a Special Interest Group (**SIG**) an operate this group with open source organisational patterns. E.g. split the responsibilities in that group into committers (people with commit rights) developers (people who submit pull requests) and follow for example the patterns used by large open source bodies like apache (https://www.apache.org/foundation/how-it-works.html) or other well established open source communities.

Usually these groups formally meet once a week and keep their own backlog.



KOMMITMENT

vertical team-split - the cellular split pattern



Further patterns only loosely related to verticalization like the learning organization, attracting experts...

One size does not fit all

One of the principles that our organisation recently came up with is, that vertical teams should be colocated. Everybody agreed and it was obvious to me that verticalization is the key to distributed development with then colocated vertical teams. It is true that distributed teams suffer due to communication overhead, different cultures at the locations, time zones and many other difficulties. So thanks to my personal experience at companies like XING or Deutsche Post I know colocation is a lot easier than distribution. It would be a small step from here to the statement "**distributed teams do not work**". When I recently asked Mark * who is the tech-lead of a distributed team how we could get his team to become colocated e.g. by swapping people with other teams he said, "but why, our distributed team works just fine". So after some arguing I realized my prejudice and we tried to find out, why their distributed team works just fine. He answers were:

- we frequently meet at one location
- the distribution is somehow balanced with no major site that has most of the people
- we just made it work

So I had to realised, that although colocation is a good pattern that by trying to enforce the same pattern to every team

1. we almost lost the ability to discover something new
2. we almost destroyed a working team (never change a running system ;-))

So the learning here is, that although common practices and pattern are good to structure the organization it is also good to always rethink your own perspective and allow for difference. Because innovation is something that we all do not know yet. Otherwise we would have already implemented it, right?

The roles of management

Management has to setup and maintain this kind of organisation. They have to care for the difficult decisions on how to prioritize cross-cutting concerns and facilitate the communication and establish a fearless ambitious learning culture. They have to fill in the gaps that are not properly structured yet. And they have to team down the walls legacy- super-structures from the past. They have to avoid micro-management at all cost but yet deliver a solid operating product to the customer. They have to foster innovation. They have to secure the future of the company.

If successful, the ideal of management is to become superfluous. I always liked the statement of the captain of the HMS Enterprise (omega tau Podcast[4]), a British Navy vessel when asked, what his job as the most senior officer on board is, whenever someone goes overboard. This question was posed during a "man overboard" training manoeuvre, where the captain seemingly did nothing to help the situation. His answer war: "To do nothing is what I do." When asked why he answered: "There is always something else that could happen just now that would need my attention."

[4] https://omegataupodcast.net/277-life-and-work-on-hms-enterprise/

Success stories

The otto.de story (in german, sry) is here:
https://www.heise.de/developer/artikel/Johannes-Mainusch-otto-de-wie-die-Titanic-den-Eisberg-verfehlte-3491223.html

The NewStore story has yet to be told:

Summary

It is great to work with nerds and technicians and it is great to work in product development. Why? Because this is where rapid development takes place and where new technologies emerge. As nerds we are an elite who are extremely well paid and can choose the most promising jobs. But being an elite also comes with responsibility to make the best of it. This is why we strive for better organisations and products that make sense to a world that demands for products with sense.

This is why we decide to

1. use bikes over airplanes
2. democracy over hippo[5] rules
3. not to build the 47th million of cars but rather invest our skills to smaller more elegant footprints
4. learn to innovate our status quo rather than living in the fear of having no pension scheme

... or do we now?

Hannes

[5] hippo = highest paid person's opinion