EVENT STORMING

Alberto Brandolini

INVITE THE RIGHT PEOPLE

FACILI-TATOR

PROVIDE AN UNLIMITED MODELLING SURFACE

PLACE DOMAIN EVENTS ALONG A TIMELINE

DEVE-LOPER

DOMAIN EXPERT

AN ACT OF DELIBERATE COLLECTIVE LEARNING

WARNING! IT MAY BECOME ADDICTIVE

SILO BOUNDARY CROSSED

ALTERNATIVE MODEL EMERGED

DEEPER INSIGHT ACHIEVED

ENTERPRISE MODEL SKETCHED

VALIDATE IT!

WRITE A PROTOTYPE

ASK MORE QUESTIONS

VISIBLE REPRESEN-TATION OF OUR CURRENT UNDERSTANDING

Introducing EventStorming

An act of Deliberate Collective Learning

Alberto Brandolini

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

This version was published on 2021-08-26

Contents

1. Preface - 60%

I started writing this book a few years ago. It feels weird to put this sentence down because, at that time, I couldn't imagine what was going to happen. EventStorming was a few blog posts and presentations, a small community of practitioners, and a malleable tool that was still full of surprises.

I embraced the "LeanPub philosophy", publishing before the book was finished, and started an interesting ride: blog posts and presentation can be intentionally ephemeral, but a book is supposed to be here to stay and to provide reliable content for the readers to come, and not only for the current listeners.

At the same time, I was still exploring the technique and discovering new ideas and format, and the new ideas deserved a place in the book too! The information was growing as a travel diary, while I needed to keep the whole consistent. The writing was easy, but the rewriting is a completely different matter.

In the meanwhile EventStorming grew and led me to practice and explore EventStorming in many different domains around the world. Obviously, there was a price to pay for that: the time available for writing shrank between business trips, workshops, training classes, conferences and meetups. My updates on the book manuscript became less frequent, while the number of readers was steadily growing.

On the other hand, the practice progressively consolidated: some experimental techniques were now proven effective, and the book could be a little less naive than the initial writing. It just needed good quality time to be finished. Or so I thought.

The 2020 pandemic gave a severe blow to my intention to be future-proof. Every single page I wrote before March 2020 embraced the idea that EventStorming is primarily a human activity to be practiced putting everybody in the same room. But in 2020-2021, the notion of filling a closed space with people started having a sinister sound in a world where social distancing became the norm.

So, I had to face a new dilemma: *"Should I rush to finish the book proposing a tool that now looks like the old world, or should I instead modernize the content to fit the changed reality?"*

I chose the second option, even if it meant painfully starting to review and rewrite much content. Ideally, this should give you a book with good ideas for any of the possible outcomes of the pandemic: back to normal, including the travel and logistic frenzy, or stuck into an endless chain of video conferences while working from home. I have no crystal ball, but I'll try to give you options when I can.

In general, large scale formats, such as Big Picture, are the ones that most benefit from human interactions, while smaller size Process Modelling and Software Design better tolerate the constraint of remote connection.

Who is this book for

EventStorming became a way more general-purpose tool than I anticipated, spanning from software development to organization design.

My background is in software development, passing through agile consulting then becoming an entrepreneur. And ideally, this is the audience I am talking to: business people that want to leverage the potential of technology and experts that want to be engaged in meaningful projects.

Notation

I'll be strict in being consistent with my notation, using a rigid color scheme (orange for *Domain Events*, magenta for *Hot Spots*, blue for *Commands*, and so on). It is born on the available colors for sticky notes, and interestingly digital modeling platforms decided to be compatible with paper-based color palettes, so I didn't change much.

Acknowledgments

[FIXME: this is *still* going to be a big mess. So many people to thank…]

How to read this book

The current state is: there is plenty of valuable content in the book, but also much work in progress. * unfinished chapters; * finished chapters that now look a little obsolete in pandemic times pandemic; * redundant content; * I still don't know how to *end* this book.

Progress counter on top of chapters

I've added a progress indicator in the chapter headers, to provide a feeling of the completion status before reading it. The counter is arbitrary, but some readers found it useful.

Feedback

Feedback was one of those missing details in the instructions about how to write a book. I probably got too much feedback in given moments; then, my reaction was to shut off. There are a few thousand readers worldwide, and having a conversation with each one of them is not a viable option.

I just realized that in the previous version of this preface, I wrote: *"you can actually run a Denial of Service attack on me. It's probably fun!"* having experienced that, I can tell you it's not as fun as I expected.

The official feedback page[1] is on LeanPub.

There used to be a Trello Board, but I found myself not following it much, so better avoiding public references to stale information.

[1]https://leanpub.com/introducing_eventstorming/feedback

2. What does EventStorming look like? - 85%

You may have heard about EventStorming, this odd-looking workshop with massive consumption of orange sticky notes. You may have even practiced it with some colleagues in your company. And maybe you are reading this book looking for guidance about what's the best way to run an EventStorming workshop.

It's only the second paragraph, and I am already sorry to disappoint you: there is no *'best way'*.

In practice, there are several ways to run an EventStorming, with different purposes and rules, but with a few things in common.

Here you'll have a taste of what you might expect, with four different EventStorming stories.

Challenging corporate processes

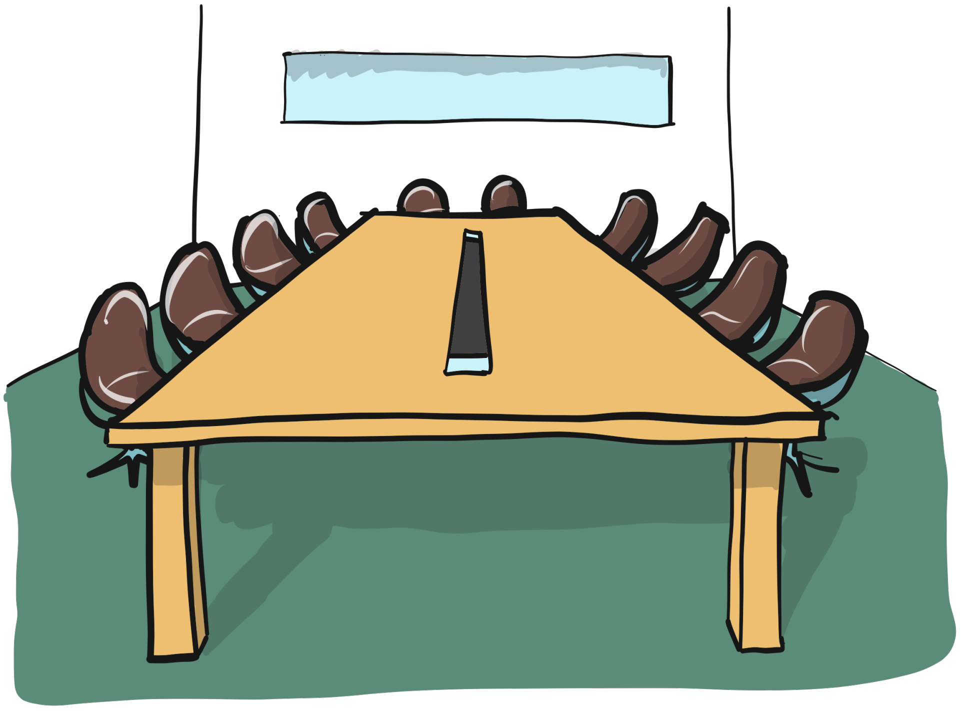
A mid-size company is struggling with a typical growth problem: the IT infrastructure that used to provide a competitive edge in the early days is slowly becoming a burden, making daily operations painful and ultimately making some strategic growth options non-viable.

I am invited to provide some big-picture level assessment of the state of the art in order to highlight the next critical actions.

The workshop

The situation looks so compromised we can't even manage the invitation process: some key people declared they wouldn't show up. However, we need to start from somewhere. We agree to call the meeting anyway, with the possibility of a follow-up.

The room looks like the typical corporate meeting room: a large table in the center and a dozen chairs around it. Poisonous[1].

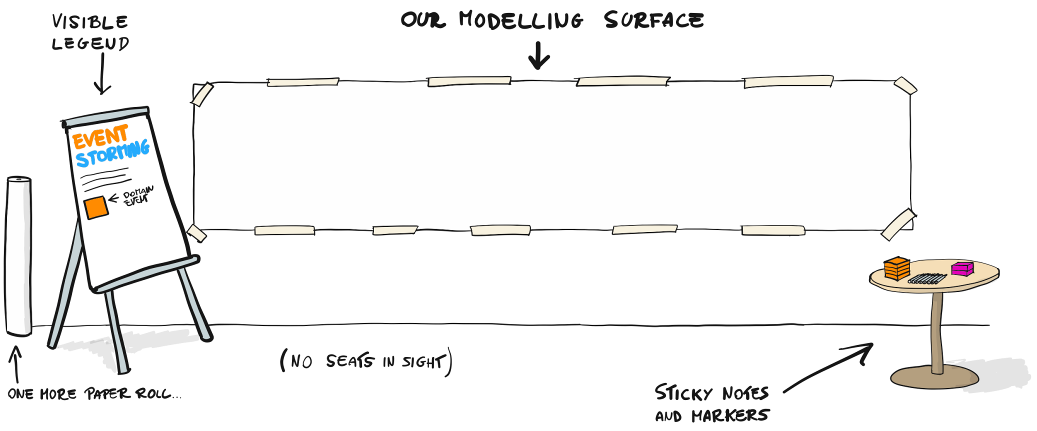


Nothing smart will ever come out from this setting, but …please, take your seat!

We take control of the room 30 minutes before the meeting's official start to hack the space in our favor. We push all the chairs to the corners and move the table to the side (it's one of those enormous meeting room tables that makes you wonder *"How the hell did they bring it in?"*), and we stick 8-9 meters of a plotter paper roll onto the main wall. Then I extract a dozen of

[1]If you want to know why, have a look to Big table at the center of the room in the Anti-patterns section.

black markers from my bag and a stockpile of sticky notes, mostly orange.

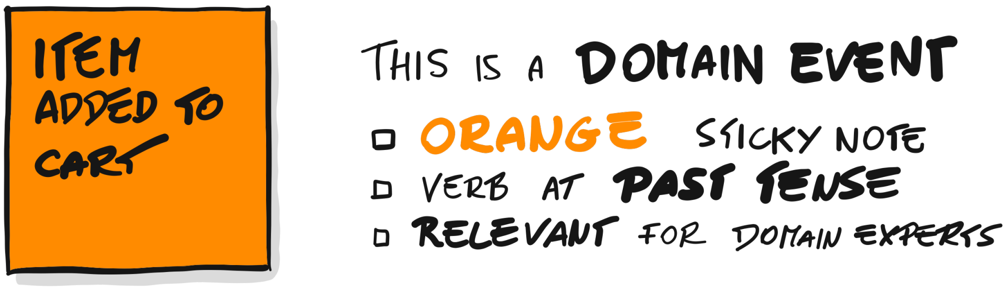


The room setup, right before the workshop

Eventually, participants - representatives from the different company departments, plus IT - start to show up. Some look puzzled at the unusual room setup, desperately looking for a seat. Someone else is still missing, but we decide to start.

I kick off the meeting with a short introduction: *“we’re asking you to write the key events in your business context as orange sticky notes, using a verb in past tense form, and to place them along a timeline”*.

Given the time constraints(2-3 hours overall), we agree to narrow the scope to the process of customer acquisition.



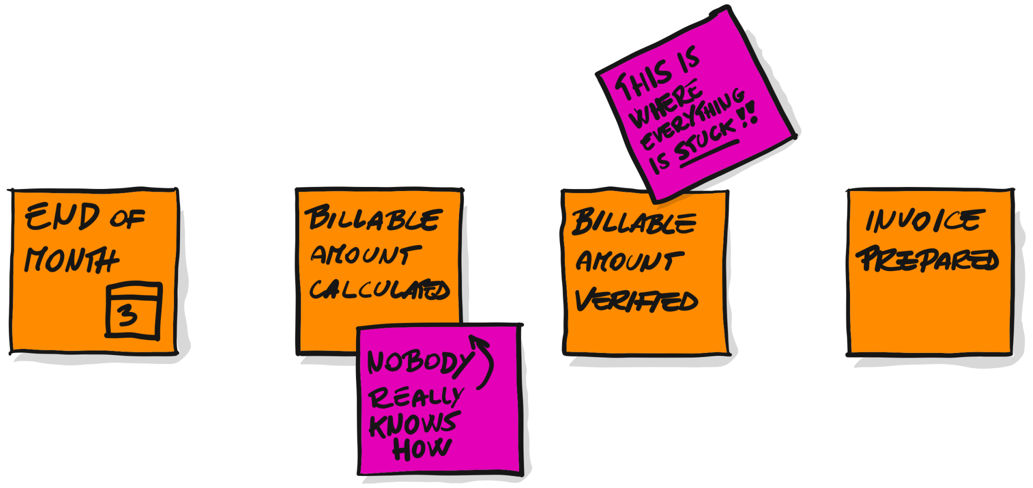
All you need to know about a Domain Event to get you started

Some participants still look confused, but most start writing their stickies and placing them on the paper roll. The *icebreakers* are doing their job brilliantly, acting as examples for the colleagues that quickly begin to imitate them. This way, we're getting into a state of *flow* in a few minutes.

I offer my help to those who still look uncomfortable, answering questions or providing a new marker if they're missing one. It turns out that I misunderstood their disengagement: *"I started working here two weeks ago. I finally understand what this company is doing!"*, a person whispers.

Eventually, the whole process starts to make sense, and while we're merging the different points of view, people can't stop commenting on given business steps: *"and this is where everything screws up!"* or *"this never really works as expected"* or *"it always takes ages to complete this step"*.

These pieces of information are crucial, so I try to capture every warning sign and write them on purple stickies decorating them with big exclamation marks, then I place them close to the corresponding step of the emerging workflow.



Capture warnings, discussions, and question marks along the way

As we expected, the number of critical issues is significant; but placing them close to the corresponding events highlights where we should concentrate our efforts.

Even my mood is different now. I asked very beginner's questions initially, but now I feel that we're getting close to the core problem quickly.

From a business point of view, the visualized process looks sound and reasonable, except for a couple of steps whose meaning I can't still figure out. The domain has many hard-to-understand regulatory constraints and bureaucratic steps, but these ones still look *really* awkward to my fresh eye.

It turns out that the steps *are* awkward. The software is imposing a couple of extra process steps to guarantee consistency of the internal data structure, but this approach is unnaturally pushing complexity outside the software[2], forcing users to do extra manual activities on paper and phone.

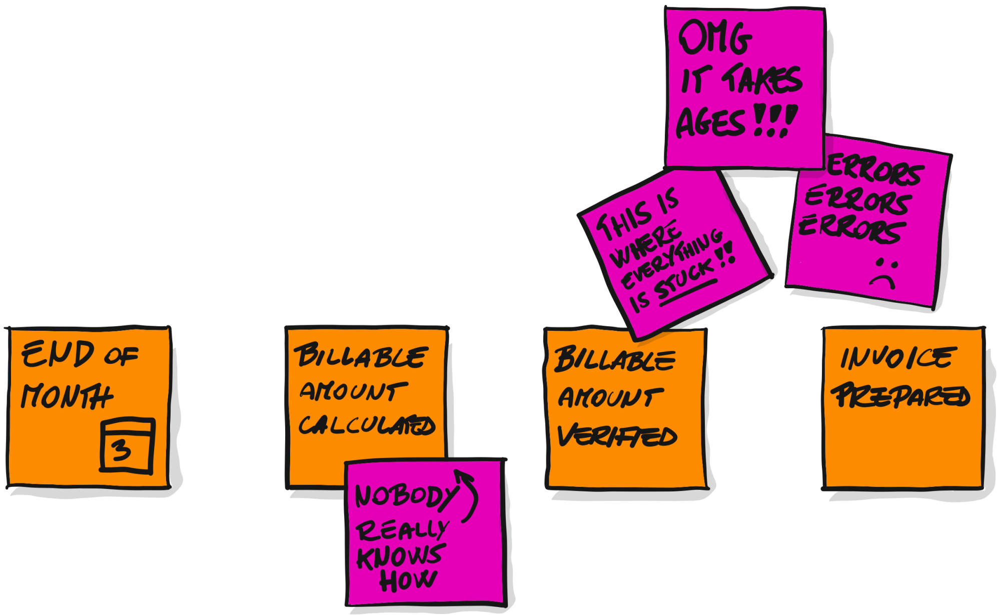
Once explored and visualized, the resulting real-world process looks bumpy and error-prone. With a disappointing consequence: the process ended up *postponing customer acquisition*, not the smartest move in a competitive

[2]This is a pretty recurrent modeling anti-pattern, I named it *hypocrite modeling*. We'll talk about it in the modeling in the large section.

market.

Luckily, *we invited the right people*: decision-makers are in the room. They propose a variation over the existing process on the fly, relaxing constraints for mandatory documentation. The currently applied requirements are unreasonably strict, so getting rid of their unintended consequences is an easy shot.

Time is getting short, so we ask if all the current blockers are currently displayed in the process (we added a couple more). Then we ask participants to prioritize them. They almost unanimously point towards one. Easy.



Looks like we have a clear winner. This is where we need to focus.

It looks like we have a clear mission now.

Aftermaths

In the afternoon, the next action becomes obvious: since we highlighted the main blocker in the flow, there's nothing more important to do than *solving that problem* (or at least trying).

Finding a solution won't be as easy as spotting the problem. And the blocker is only the visible symptom, not the root cause. But *choosing the right problem to solve is a valuable result* and we achieved it *quickly*.

There's a feeling of urgency to prototype a solution. So, after taking pictures (in case someone tears the paper roll down), I start working on an alternative model for the selected *hot spot*, together with the software architect, based on two separated models instead of a single bloated one.

Even if we established a clear link between the main process blocker and a possible solution, putting it to work is not only a technical issue: permissions needed to be granted, and roles and responsibilities in the organization around that step needed to be redesigned to overcome the current block.

Despite the initial assumption that software was the most critical part, it turned out that the solution was mostly *politics*. And when it comes to politics, transitions usually take more than expected, and they're never *linear* or *black-and-white*.

Kicking off a startup

Founders of a new cool startup are hiring a cool software development company and a cool UX agency to write their cool backbone software.

The business model is new for the market. It's all clear in the founders' brain since they are very experienced in the financial domain. The development team is already familiar with Domain-Driven Design concepts, but developers, the ones supposed to make the thing real, have no previous domain knowledge - excluding some regrettable experience as customers - and the whole thing looks *huge*.

I am invited to join as a facilitator for an exploration workshop: the goal is to have the development team up to speed, learning as quickly as possible about the domain. Moreover, founders know about the business domain but are unaware of the technical risks hidden at the implementation level.

Given the number and complexity of mandatory compliance constraints in the domain, the Minimum Viable Product[3] is going to be relatively big, compared to the ideal startup Eric Ries's style. Anyway, we'll need to keep under control the natural tendency to inflate the MVP since hitting the market as soon as possible is critical.

Day one

At 9 am, eight people are in the room: representing the business, technical, and UX perspectives of the exploration.

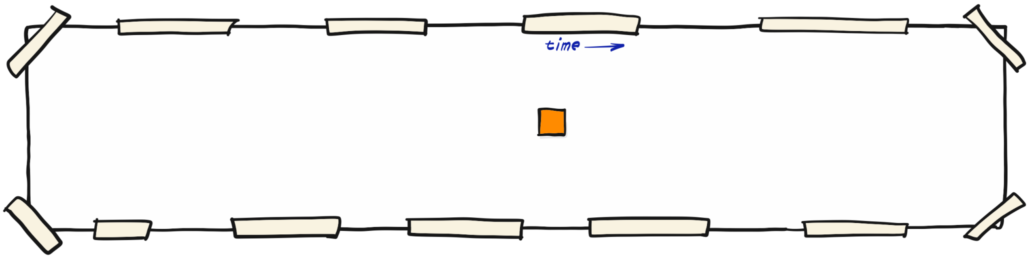
During preliminary introductions and coffe, I lay down my secret weapons: I unroll as much as I can of a plotter paper roll and stick it to the wall, then extract some hundreds of orange sticky notes from my bag and put them on the table, together with a ridiculous amount of black markers.

Some people already know about EventStorming, but the founders are totally new to the concept. However, they like the idea. The unusual setup already triggered curiosity, so I don't have to spend much time introducing the workshop. Instead, I immediately ask everybody to write the key *Domain Events* of the business process on orange sticky notes and place them according to a timeline on the paper roll.

The request sounds somewhat awkward because we're skipping introductions and explanations. What we really need is an example: once the first

[3] A *Minimum Viable Product* is one of the most intriguing and debated ideas presented in Eric Ries's book *The Lean Startup*, a *must-read* for every startupper in the digital world. In order to gather feedback from your potential customers as soon as possible, an MVP is the smallest possible product that can be shipped. The goal is to avoid wasting money building features, only to discover that no one would buy them.

Domain Event is in place, everybody is more confident about what a Domain Event can be.



Gotta start somewhere…

We quickly enter a state of *flow*: once the first Domain Event is placed on the timeline, it becomes fairly obvious what has to happen *before* and what happens *after*, so everybody starts contributing to the model.

Some orange stickies are duplicated: probably somebody had the same idea, others look alike but with slightly different wordings. We keep them all, for the moment. We'll choose the perfect wording later, once we have some more information available.

I also remark that *"there is no right one"*. Looking for the perfect wording will slow us down in this phase.

A few stickies do not comply with the Domain Event definition I provided: there's some sort of *phase* instead of a *verb at past tense*. But *phases can hide complexity* from our investigation, so I just turn the sticky note 45\▯ anticlockwise to signal that something is wrong, without disrupting the discussion flow.

This is not a Domain Event, this is a process…

The modeling style feels awkward but powerful. We try to model processes before the detailed explanation from the domain expert.

There is no description of the tedious, *easy-to-guess* steps of the business process. But every time the developers are off-target, then a meaningful conversation arises. We don't talk about *"the user needs to login in the system"*, (*boring*); instead, we talk about places where there's more to learn.

It's more *discovery* than *translation*. This way, it feels like developers are genuinely getting into the problem.

Conversations, conversations, conversations

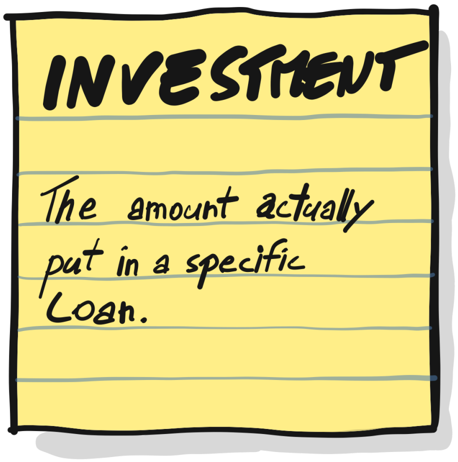
Sometimes, the founders provide us with really cool insights about the domain's unexpected complexity. We listen like pupils in a classroom, taking notes, formalizing the just learned complexity in terms of Domain Events and adding more precise definitions to the vocabulary.

When new terms arise, and the discussion shows that they have an exact meaning in that *context*, I start capturing key term definitions on a special sticky note and place them just below the normal flow.

It won't be a *Wikipedia-ready* definition: just the precise meaning of that term in that specific conversation.

Sometimes the knowledge is on the technical side: a developer takes the lead explaining the complexity lying behind an apparently simple problem.

We could actually see the business and the technical parties getting wiser every round.

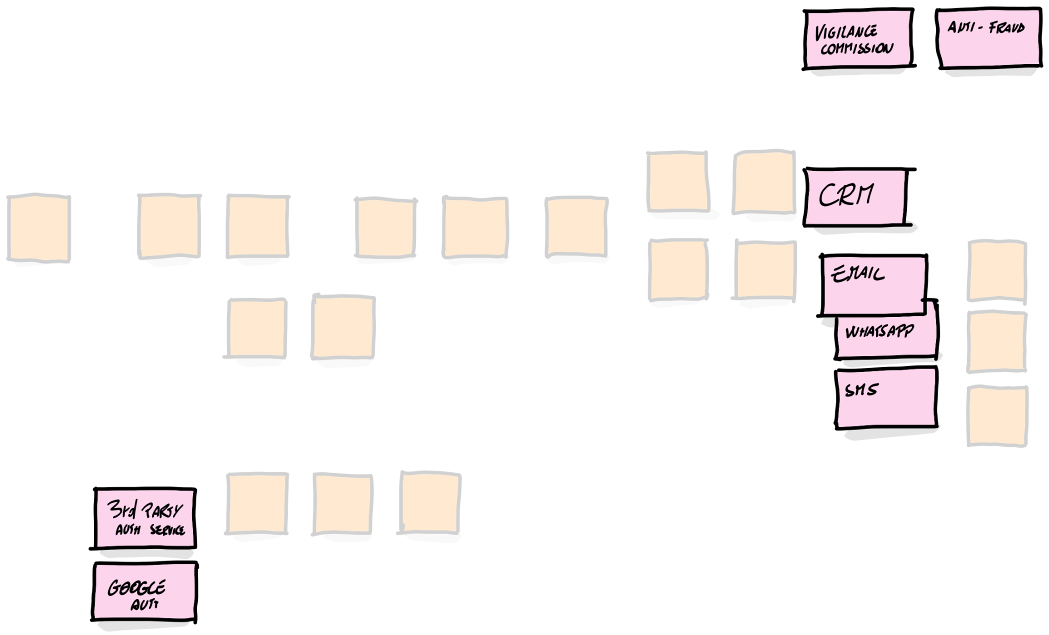


When precision emerges from the words of a domain expert, better write it somewhere…

Open-ended conversations are my favorites. On a couple of occasions, the founder candidly says: *"I have no idea. We need to explore this scenario."* I loved that. Honest domain experts admitting they don't know something are a million times better than a *wanna-be-domain-expert* mocking up answers to stuff they have no clue about.

While people keep exploring different corners of the domain, I start annotating all the hot spots in purple stickies. They may represent *problems*, *risks*, *areas that need further exploration*, *choice points* where we don't have enough information yet or portions of the system with *key requirements*.

Eventually, we start mentioning *external systems*: they can be external organizations, services, or online apps. I start representing with larger pink stickies the external systems our cool new software will have to deal with. This will happen more explicitly later on, but having some examples already available will make the following steps easier.



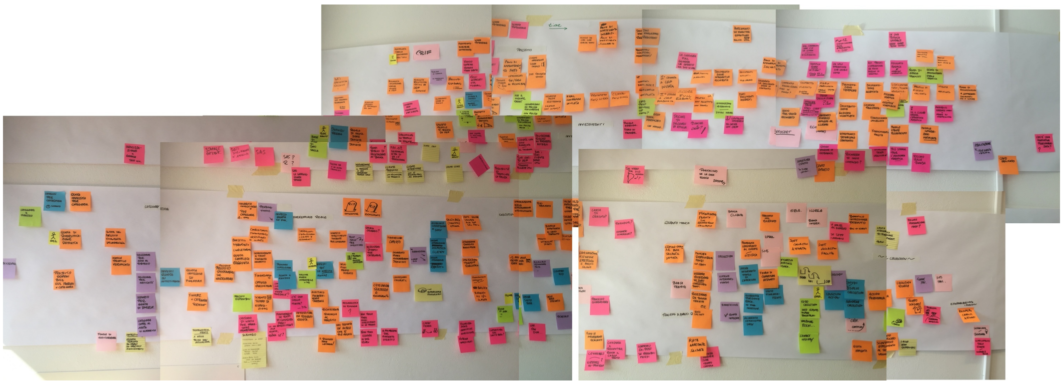
External systems started to appear quickly. For each one of them, we wrote a larger light pink sticky note.

As a background process, I start to label specific areas of the business flow[4]. Something like *Customer Registration*, *Claims* or *Fraud Detection*. We're starting to see independent models that could be implemented independently, with different, relatively well-defined purposes.

I am not expecting precision and clean boundary definitions at this stage, just some emerging structure that we will improve later.

It turns out that there are already a few solutions available off-the-shelf for given portions of the flow: no need to develop them in-house unless we need to do something distinctively different.

[4] I have a special tool for that, a removable label that can be moved around just like a sticky note. I'll talk about the tooling in the Tools section.



Too big to fit into one picture, but still manageable.

At the end of the day, we filled up 18 meters of process modeling. A single wall isn't enough to contain the whole stuff, so we used the opposite wall too. We acknowledge that there were some more areas to be explored, but the overall picture of the process looks solid.

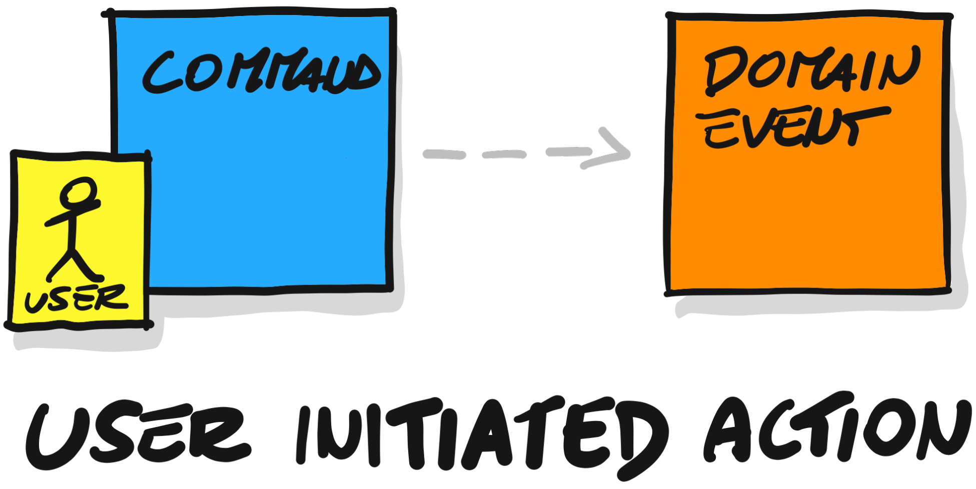
After contemplating our artifact, we head for dinner with the feeling of having accomplished a lot.

Day two

We start the session by reviewing the model we left from the day before.

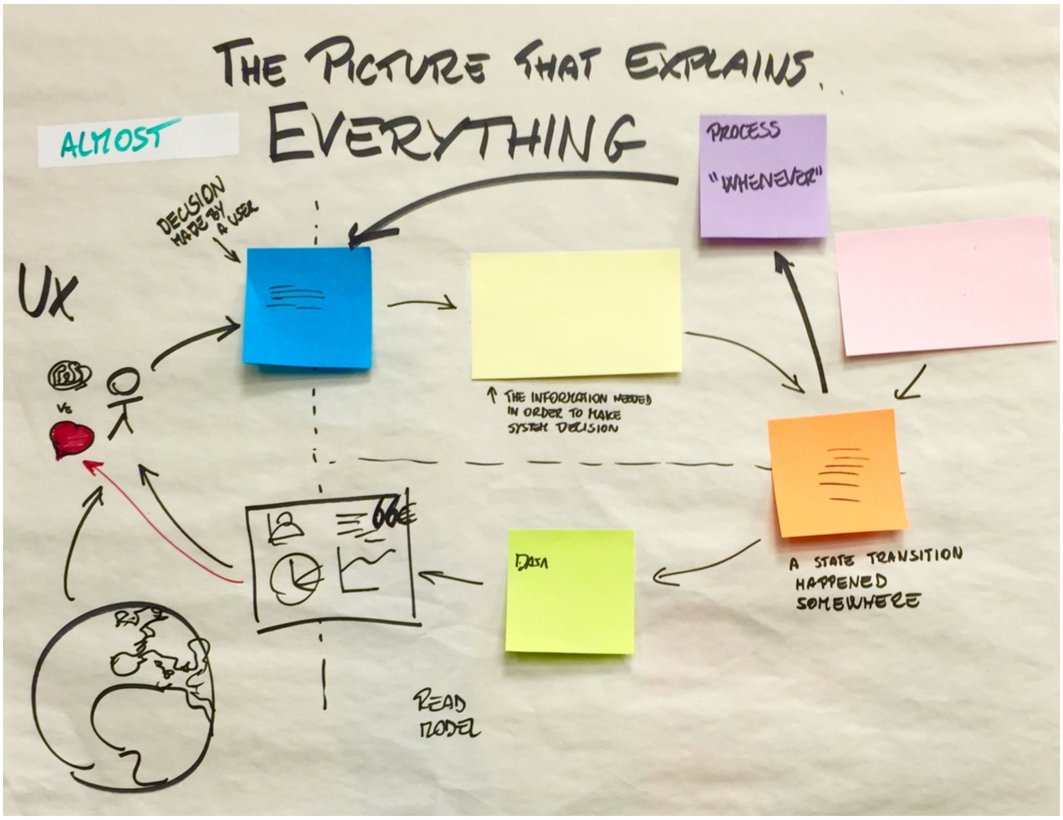
Some stickies now look naive compared to our new level of understanding. We rewrote a few with more precision and discarded some of the purple question marks that were covered in yesterday's session.

It's time to get a deeper look into the mechanics of the core components of the system: we start introducing more rigor in our process by introducing *Commands* representing *user intentions/actions/decisions* (they are blue stickies where we write something like **Place Order** or **Send Invitation**), and *Actors* (little yellow stickies) for specific user categories.



Adding the little yellow sticky can make the difference in shifting the focus towards user interaction

To provide a little more background, I draw *The picture that explains everything* (see chapter The picture that explains everything for more on that) on a flip chart and keep it as a reference.

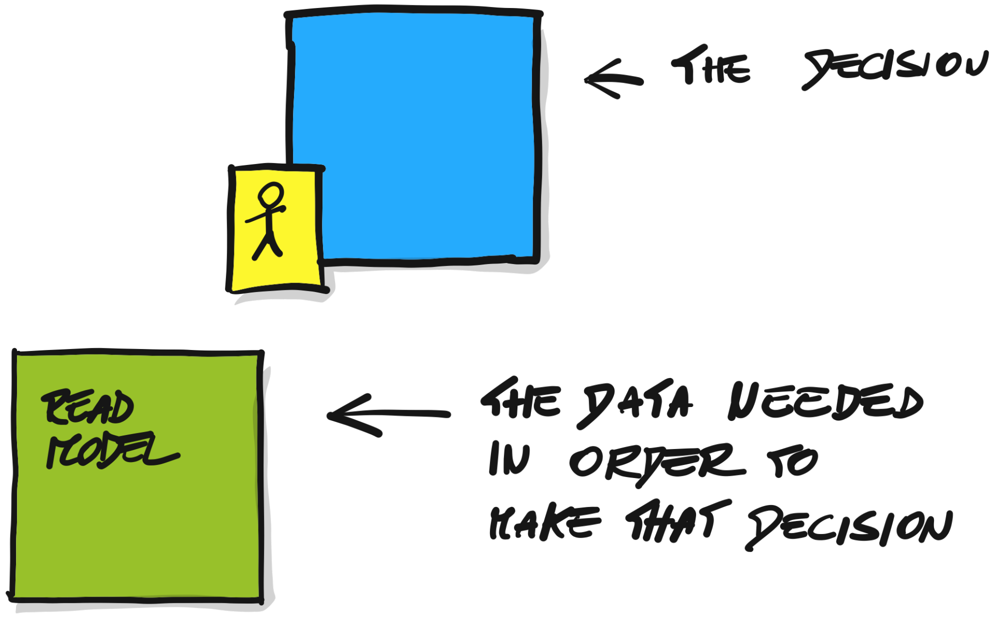

THE PICTURE THAT EXPLAINS...
EVERYTHING
ALMOST
PROCESS
"WHENEVER"
DECISION MADE BY A USER
UX
VS
THE INFORMATION NEEDED IN ORDER TO MAKE SYSTEM DECISION
A STATE TRANSITION HAPPENED SOMEWHERE
DATA
66€
READ MODEL

The incredibly ambitious "picture that explains everything"

Adding commands and specific actors triggers more discussions about the reasons why given users will perform given steps. Once more, some steps are trivial, while a few of them trigger deeper discussions like: *"Why should user X activate the service?"* which end up challenging the original assumptions.

A couple of interesting things happen around commands and actors. Commands are ultimately the result of some user *decision*, but thinking in terms of user decisions (*Can we make it easier? Can we make it better? Can we influence it?*) forces us to think in terms of the relevant data for a given decision step.

We capture this information in *Read Models* (green stickies in our colored dictionary), or in little UI sketches if visual are relevant for the current step, like in *"we need an image here"*, or *"price needs to be displayed in the center of the screen"*.



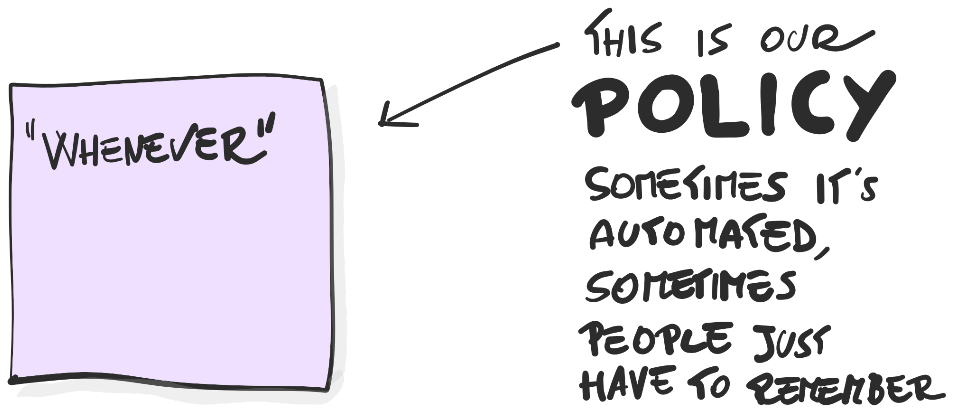
Read models are emerging as tools to support the decision-making process happening in the user's brain

Interestingly, while exploring the users' motivation, we realize that not every user in a given role thinks in the same way. Even if the decision might be the same, the reasons behind it will vary a lot.

What used to be a simple *user* or *actor* turns out to be a more sophisticated collection of *personas*. Since we're in Lean Startup mode, it is nice to see this concept emerge: we might eventually use it to sharpen our MVP by focusing only on few selected personas.

However, we don't care about the exact definition. The fact that *we're having this conversation* is way more important than the precision of the used notation.

The more we look into local mechanics, the more we find ourselves thinking in terms of *reactions* to specific Domain Events. It's easy to pick them up since they mostly start with the word 'whenever'. It's something like *"whenever the exposure passes the given threshold, we need to notify the risk manager"* or *"whenever users log in from a new device, we send them an SMS warning"*. Lilac is the color for this reactive logic that takes place right after an event and triggers one or more commands somewhere else. 'Policy' is the name we end up using in this specific workshop.


"WHENEVER"
THIS IS OUR
POLICY
SOMETIMES IT'S AUTOMATED, SOMETIMES PEOPLE JUST HAVE TO REMEMBER

Our policy, regardless if it's manual or automatic

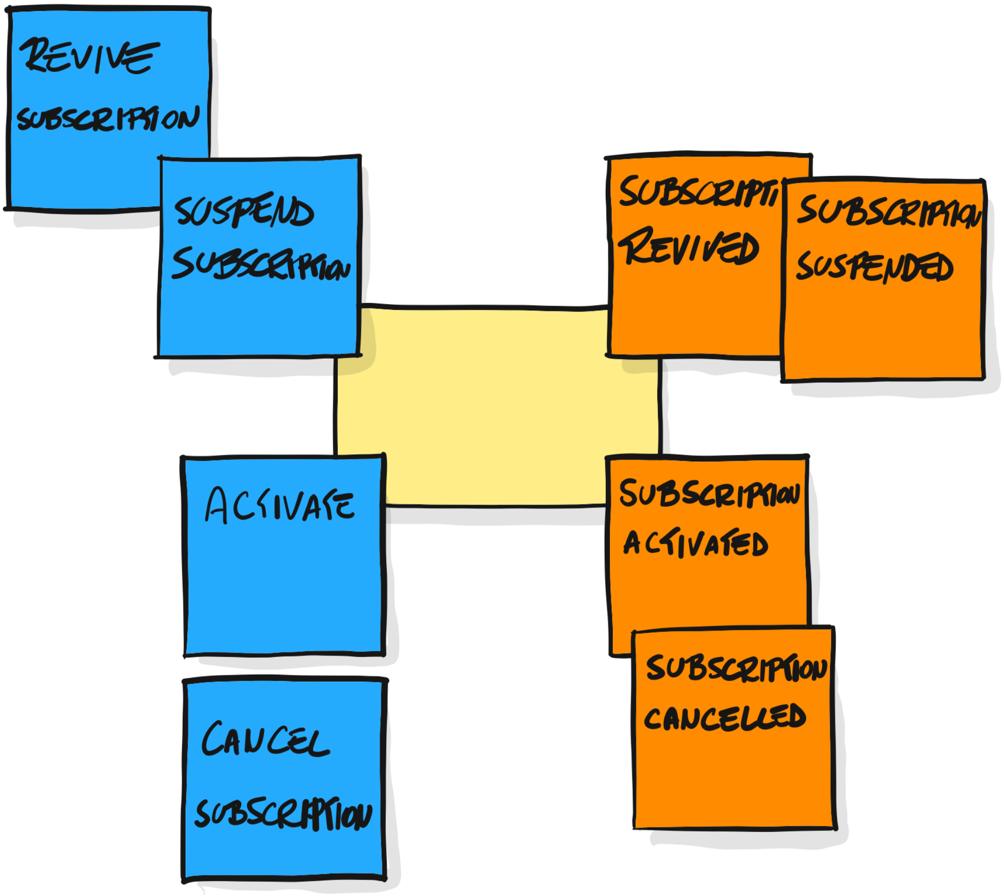
It's interesting to capture policies as early as possible, without making assumptions about their implementation. In a startup, some actions will have to be implemented by a system one day, but it's usually better to defer commitments (and the related expenses for licenses and fees) to the moment the customer base will be big enough to justify the investment.

A manual and possibly cumbersome process today will eventually become a software solution tomorrow. This way, we can manage growth without precise information, market feedback above all.

Developers are getting excited because "we're getting technical" and an Event-Driven architecture is emerging from the mass of stickies. But business people aren't excluded from the conversation: the reference notation is still visible on the flip chart acting as a visual legend, and we're mostly discussing the whys or providing useful insights on the tricky areas.

Eventually, while founders move to a different office, we keep exploring the model with developers. They were eager to get into technicalities like cats smelling fish.

I agree. This is the time. Key *Aggregates* (in the traditional pale-yellow color) start to pop-up, in a responsibility-driven fashion. Given the bounded contexts have already been sketched, most aggregates look somewhat trivial.



A very simple aggregate, apparently

Interestingly, the exploration feels complete, even if it's process-oriented. We had pretty much everything sorted out without using words like *database*, *table*, and *query*. Data comes in mostly in terms of mandatory requirements or as a read model required to support a given user step. Only a few places still look like CRUDs, but most of the business is described in terms of *Events* flowing in the system.

And it feels so right.

Finally, it looks like we've covered pretty much everything. What looked like an unholy mess at the beginning now looks like a reasonable flow. The team confidence is sky-high. There are a couple of areas where experiments will be necessary (mostly because we need to understand system dynamics after hitting a considerable amount of users), but the overall feeling is that development would be on track. Everybody knows the reason why given pieces are there, and a lot of typical mistakes and misunderstandings that will look stupid in hindsight just won't happen.

We are in *"the startup's twilight zone"*: assumptions should be challenged with experiments, but choosing the right ones to challenge and designing the corresponding experiment is still mostly an art.

Designing a new feature for a web app

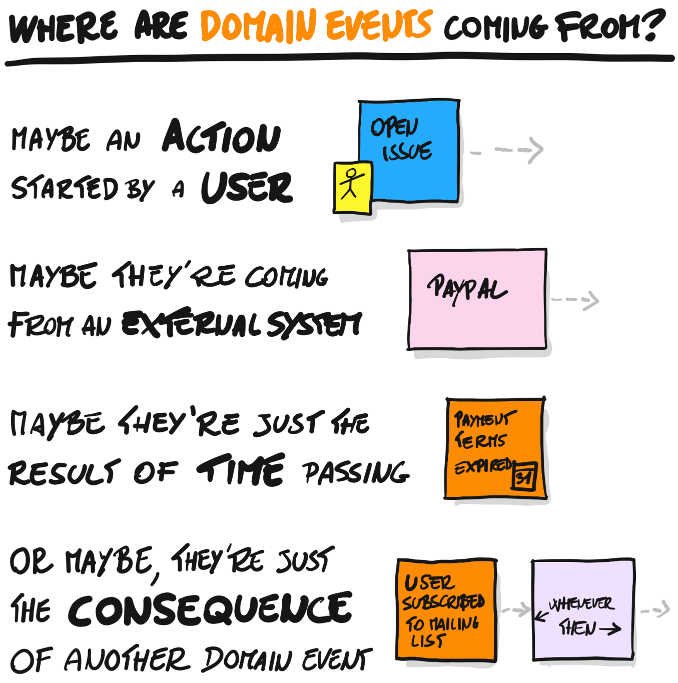
We just run a big picture workshop in the morning, but now it's time to narrow down the scope and focus on the new requirements for our app.

We don't need every domain expert here: the scope of the new feature is relatively clear and involves only two of them.

[FIXME: probably a little later] With the development team, we dig deeper into the semantics of the Domain Events.

It's funny to see how many times we're *rewriting* the sticky notes with different names, exploring variations or adding more precision to the picture. Even if we're throwing away many stickies, we still have a feeling of progress and increased solidity.

At this moment, we start getting deeper into the mechanics of our business process: every Domain Event should be the result of something. I sketch something similar to the picture below on the visible legend.


WHERE ARE DOMAIN EVENTS COMING FROM?
MAYBE AN ACTION STARTED BY A USER
OPEN ISSUE
MAYBE THEY'RE COMING FROM AN EXTERNAL SYSTEM
PAYPAL
MAYBE THEY'RE JUST THE RESULT OF TIME PASSING
PAYMENT TERMS EXPIRED
31
OR MAYBE, THEY'RE JUST THE CONSEQUENCE OF ANOTHER DOMAIN EVENT
USER SUBSCRIBED TO MAILING LIST
WHENEVER
THEN

Where are domain events coming from? Four different possible answers to the question.

- maybe a *Command* (a *blue* square sticky note) triggered by a given *User*;
- maybe the Domain Event has been triggered by some *External System* (normally a *pink* larger rectangular sticky note);
- maybe it's just the result of *time* passing, without any particular action involved (like `PaymentTermsExpired`);
- or maybe it's just the *consequence* of some other event: *whenever* one thing happens, *then* another one will happen. Well, it's not always that obvious, so we set up a lilac sticky note for that.

Some developers are puzzled. A few commands look just like the rephrasing of the corresponding domain events. It's always funny to see how developers' brain reacts to boring and repetitive tasks. In fact, that's not a problem: the whole thing is complicated, but this doesn't mean that every little brick has to be a mess. Some will be trivial.

However, in given portions of the flow, this is not happening: the relationship between an user action and a system reaction is more sophisticated than we initially thought, and can trigger different consequences.

Working with developers always brings an interesting consequence: they're wired up in thinking with alternative and complementary paths, looking for some form of *semantic symmetry*: so if we had a `ReserveSeat` command, they immediately look for a `CancelReservation` and start exploring the alternative path. This lead us to find more events and more *"what happens when …?"* questions.

It's now time to introduce *Aggregates* one of the key concepts of Domain-Driven Design, by looking for local responsibilities that could trigger different responses to commands. We use plain old yellow stickies for that.

It feels natural to start grouping commands and events around the newly found aggregates, and this triggers even more discussion in the hunt for symmetry. Aggregates now are starting to look like little state machines.

Some people are puzzled because the moment we started to group around responsibilities, we broke the timeline. That's fine, the two groupings are orthogonal: we can't have both. The timeline was great for triggering big picture reasoning, but now *responsibility* is a better driver for robust system design.

The picture that explains everything is our companion again, and I refer to it in order to introduce *Read models* in green to support user decisions and *processes* in lilac, that take care of the system reactive logic.

[FIXME: Finish!]

Quick EventStorming in Avanscoperta[5]

It's time to run a retrospective in our company. Training class sales haven't been as good as expected in the last weeks, and it makes sense to explore the problems that we surfaced along the way.

Since we advocate large modeling surfaces, we have plenty of space to model. Every available wall of hour headquarter (a single 60 square meters room) is writable, so there's no need for the typical paper roll here. However, a large supply of orange stickies is mandatory.

The company core is pretty small: only 4-5 people. However, there's a significant amount of autonomy to take process decisions, and we experimented with many changes lately, so it makes sense to merge the different perceptions of the process into a consistent whole.

Everyone of us gets a marker. We start writing *Domain Events* (short sentences like **Training Description Published** or **Ticket Sold**) on orange sticky notes and place them along a timeline, from left to right.

There is no strict consequentiality: everyone writes about the topic they know best and find the corresponding place in the flow, eventually adjusting the position of surrounding stickies when things get too packed or too clumsy.

Along the way, we realize we keep mentioning External Systems (we have the pink larger stickies for that) and something that doesn't belong to the usual EventStorming toolkit: *communities*. They are meaningful enough in our context to justify a specific notation: we pick a color (light blue) that hasn't been assigned yet and add it to our incremental notation.

When the overall flow starts to make sense, we begin exploring issues, writing them down as purple sticky notes that we place close to the corresponding domain events. In our case, we had something like **Training Class Description Sucks!** and **Unclear Discount Policy**, or **Advertising Policies!!!** to highlight that we didn't have a repeatable approach to advertising. Ideas come together, and we end up with around 20 possibilities for improvement.

[5]Avanscoperta is my small company, where I try to eat my own dog-food. www.avanscoperta.it

The outcome of our EventStorming session in Avanscoperta (one among many).

When exploring on such a wide scope, it's easy for issues to pop up randomly mixing the very important things with the trivial ones, so - just like we would do in a retrospective - we'd prioritize issues, with the impact in mind: *"Which issue is going to have the biggest impact when solved?"* We decide to focus on the appeal of the product we're selling. The decision to buy a ticket for a training class - which we'll represent as a blue sticky note - is a user decision that will involve logical and emotional thinking.

On the logical side, we have a little problem with the data displayed: though the current price is currently displayed, our naming policy for discount is somewhat confusing for the users: two prices are shown, and some users wonder whether they lead to different service classes. Maybe not a big blocker, but the buying experience looks clumsy.

The big blocker stands right in our faces: the purple sticky note saying **Training Class Description Sucks!** stands right in front of us, telling us that the perceived value of what we're selling is not enough to trigger the purchase.

A quick look at the training class description of our best sellers, compared to weaker ones, confirms the hypothesis: despite good teachers and cool topics, the class as shown on the website is not triggering any strong call to action.

Now we know that we should put a lot more effort on crafting the content of the training class: it has to speak to the rational mind, but it has to be catchy too for the emotional part of the brain (or 'the heart' if you are feeling more romantic than scientific).

In the end, my brain is torn apart. As an entrepreneur, I am quite happy I've identified problem number one on the list, and now I have an action plan towards a possible solution. As a developer, I am slightly disappointed since the solution doesn't involve writing any code.

I trust the next bottleneck to be better.

You've now got a little taste of what an EventStorming session might look like. There's something different in every single approach, and many choices are largely dependent on the context. Some similarities are visible though:

- an *unlimited modeling surface*;
- a virtually *unlimited supply of markers and sticky notes* in different formats;
- a *collaborative modeling* attitude involving all key roles and stakeholders;
- *no upfront limitation of the scope* under investigation;
- focus on *domain events* along a *timeline*;
- continuous effort to maximize *engagement* of participants;
- *low-fidelity incremental notation*;
- *continuous refinement* of the model, or *progressive precision*.

Possible formats

The discipline[6] is relatively new (my first experiment dates 2013), but there's already some distinct established format. I like to think about it like Pizzas: there's a common base, but different toppings.

Here's my menu.

- **Big Picture EventStorming**: the one to use to kick off a project, with every stakeholders involved.

[6] I had shivers down my spine when I realized I associated the word 'discipline' with the apparent chaos of EventStorming.

- **Design Level EventStorming**: digs into possible implementation, often DDD-CQRS/ES[7] style.
- **Value-Driven EventStorming**: A quick way to get into value-stream mapping, using storytelling like a possible platform.
- **UX-Driven EventStorming**: similar to the one above, focusing on the User/Customer Journey in the quest for usability and flawless execution.
- **EventStorming as a Retrospective**: using domain events to baseline a flow and expand the scope in order to look for improvement opportunities.
- **EventStorming as a Learning Tool**: perfect to maximize learning for new hires.

In the next chapters, we'll start from the Big Picture approach, the one that opens the way for further explorations.

Chapter Goals:

- a narrative about what happens in a good workshop
- a glimpse of what can be achieved
- a glimpse of the underlying mindset
- an anchoring about: "I want to get there!"

[7] Domain-Driven Design, Command-Query Responsibility Segregation and Event Sourcing. We'll talk about this later.