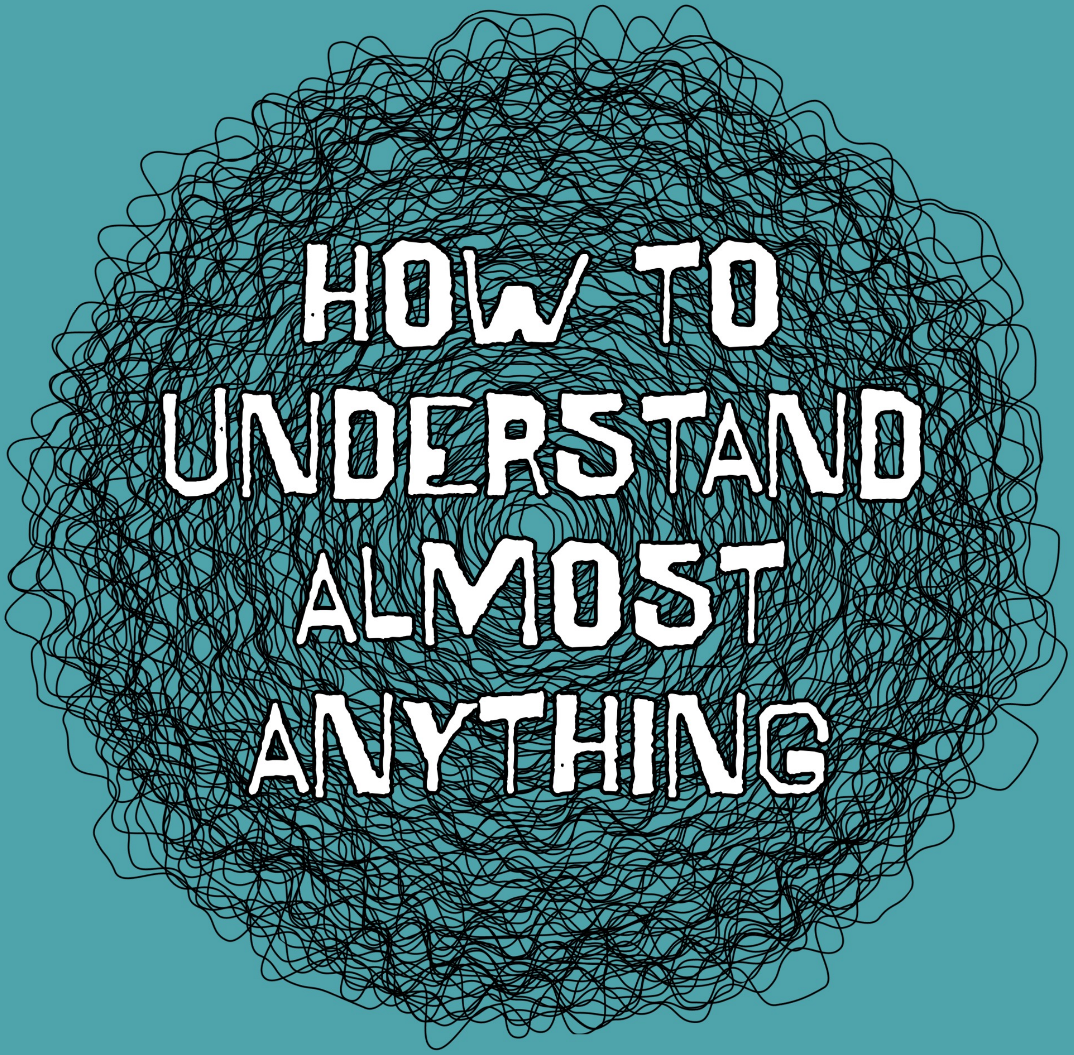
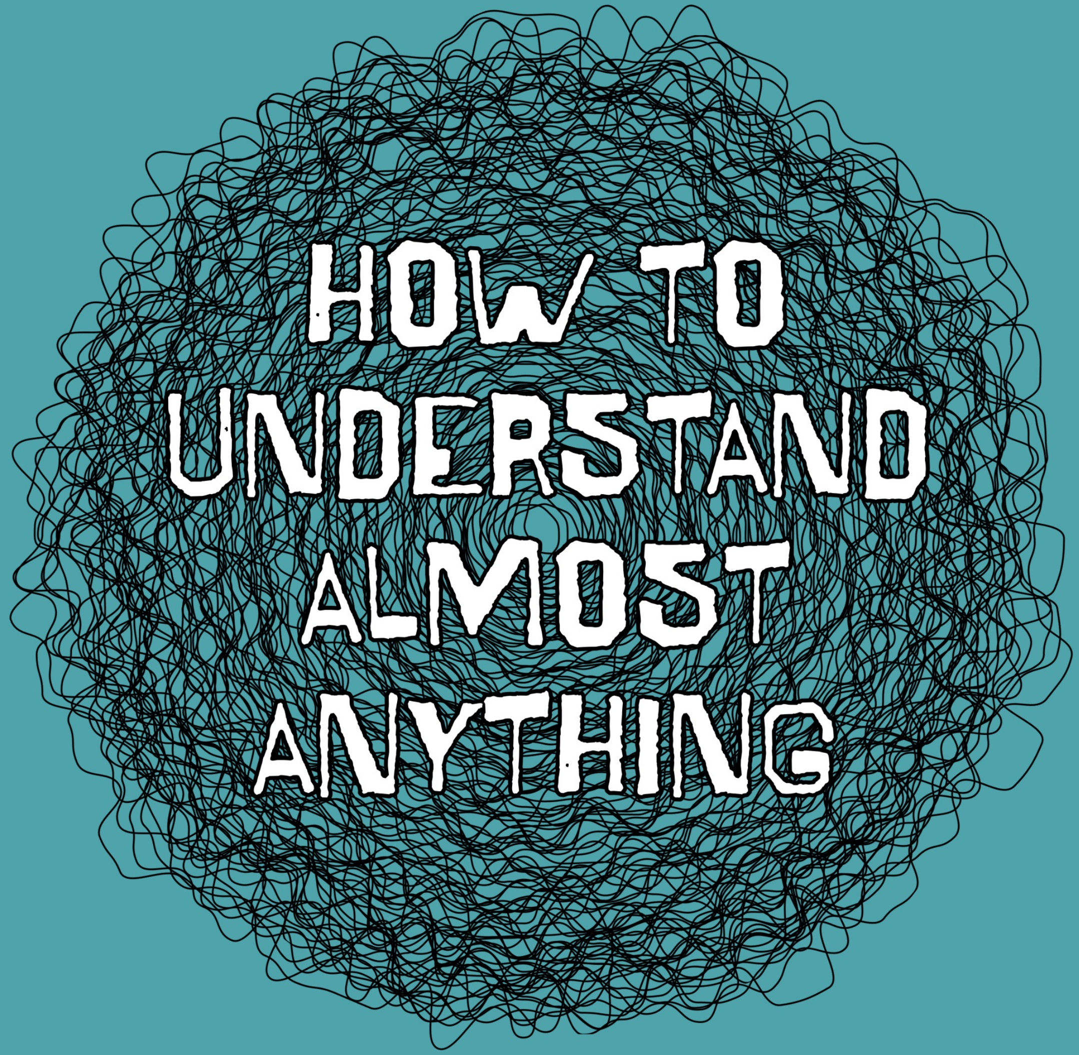
oop 2024
software meets business



DOMAIN ANALYSIS FOR PRACTITIONERS

Markus Voelter

voelter.de



DOMAIN ANALYSIS FOR PRACTITIONERS

Based on the book of the same name:

http://voelter.de/htuaa

There‘s a discount code for the PDF version at Leanpub:

https://leanpub.com/markusvoelter-htuaa/c/oop23
(expires end of Feburary)

I

INTRODUCTION

What is Domain Analysis

As the book title says:

An approach, a set of practices,
to understand almost anything.

☺

What is Domain Analysis

A **domain** is an area of interest and expertise
often owned by a particular organisation.

There are usually people who are **experts** in the domain,
they understand large parts of the subject matter.

Domain Analysis is about **capturing** this subject matter outside the brains of the experts to:

- make it accessible to a wider range of people
- make it accessible to software tools.

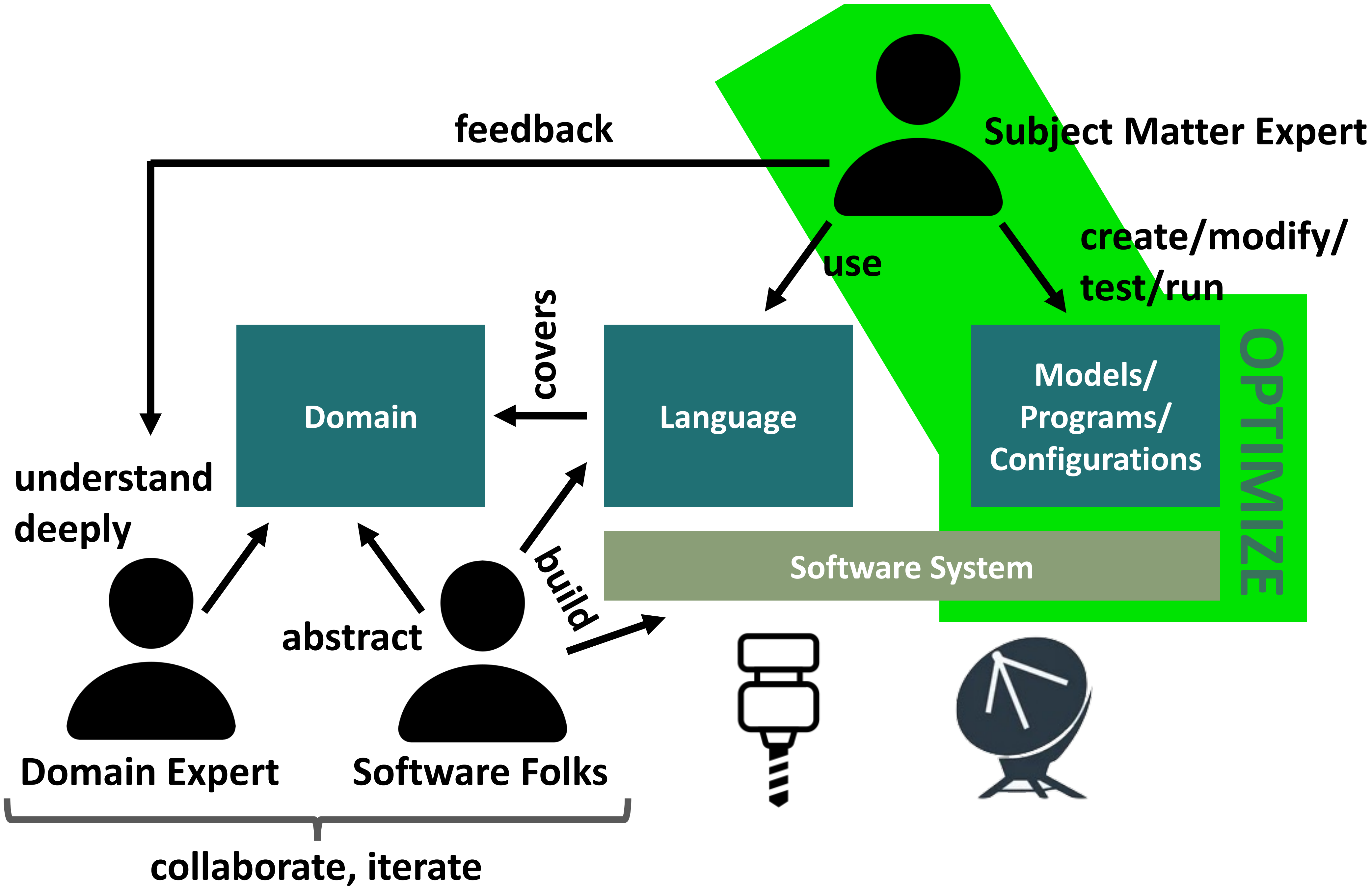
Accessible to Software Tools

- data structures
- constraints & rules
- semantics
- models, languages

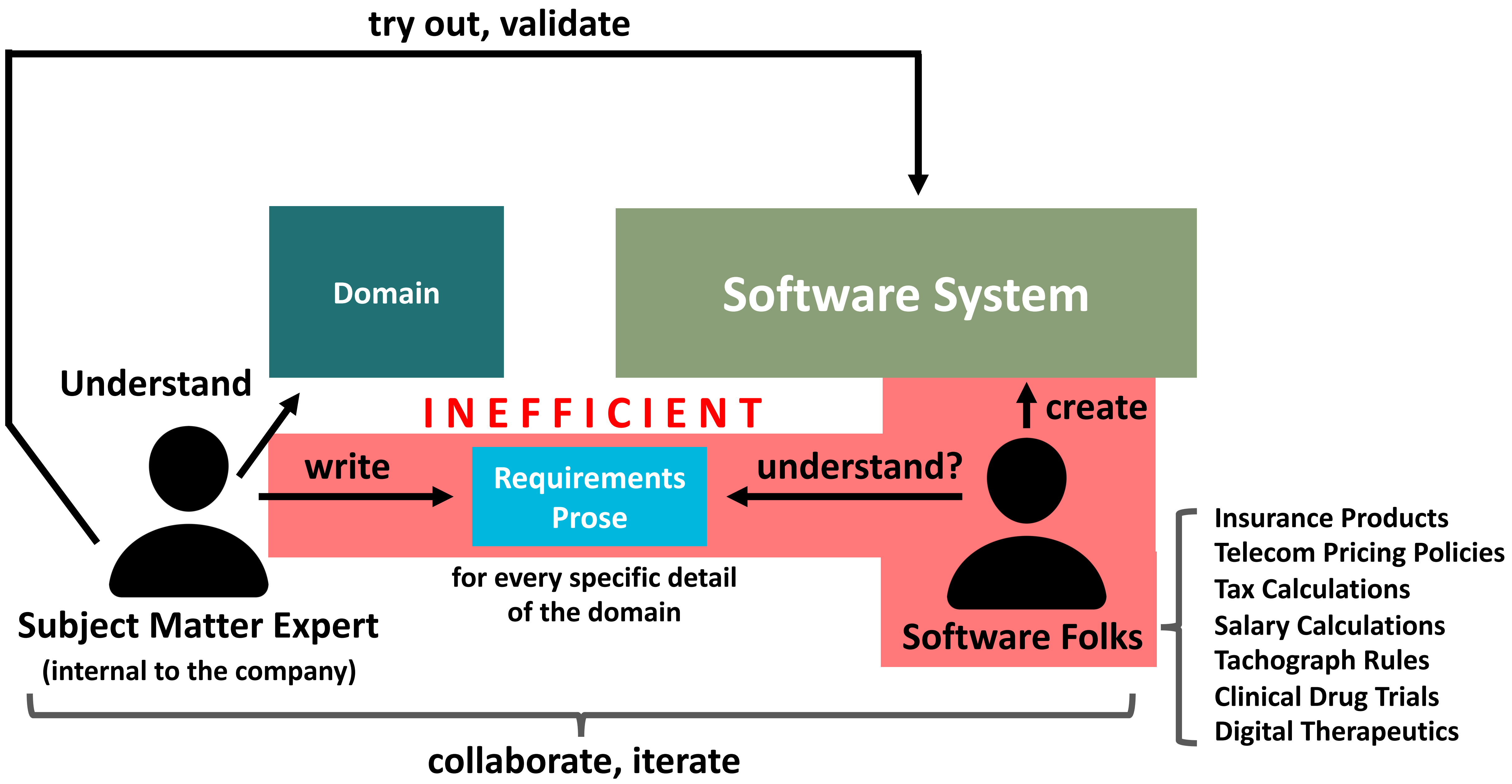
Things with properties and relationships, plus static and dynamic semantics.

- XML/Json schemas
- UML diagrams
- Meta models
- DSLs

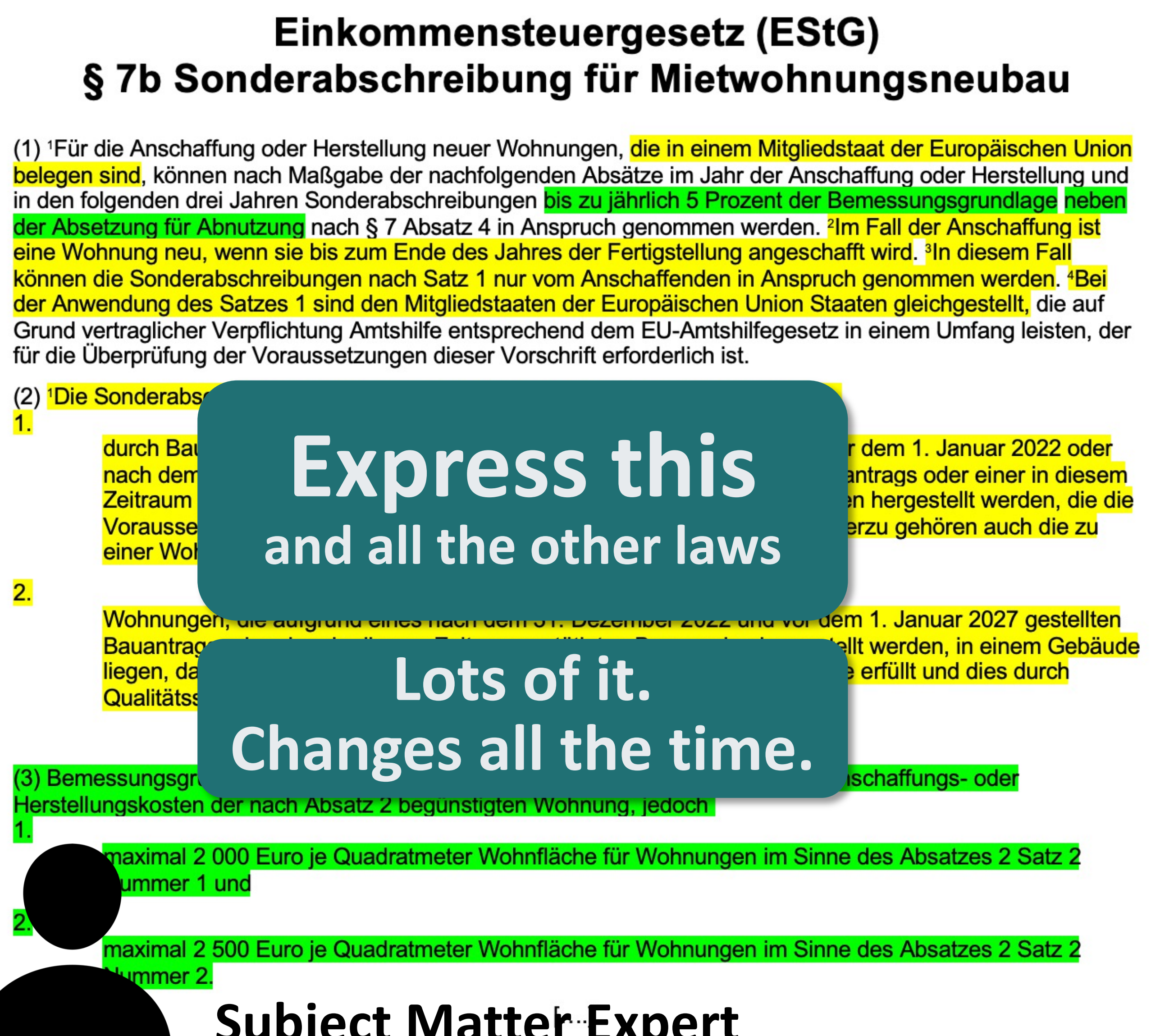
- APIs
- Software Structures
- Checkers, Compilers & Interpreters
- UIs & Simulators
- Transport Protocols


feedback
Subject Matter Expert
use
create/modify/
test/run
covers
Domain
Language
Models/
Programs/
Configurations
OPTIMIZE
understand
deeply
build
Software System
abstract
Domain Expert
Software Folks
collaborate, iterate

Which is very different from .....


try out, validate
Domain
Software System
Understand
INEFFICIENT
create
write
Requirements Prose
understand?
for every specific detail of the domain
Subject Matter Expert
(internal to the company)
Software Folks
Insurance Products
Telecom Pricing Policies
Tax Calculations
Salary Calculations
Tachograph Rules
Clinical Drug Trials
Digital Therapeutics
collaborate, iterate

Example of concrete requirements



Einkommensteuergesetz (EStG)
§ 7b Sonderabschreibung für Mietwohnungsneubau

(1) ¹Für die Anschaffung oder Herstellung neuer Wohnungen, die in einem Mitgliedstaat der Europäischen Union belegen sind, können nach Maßgabe der nachfolgenden Absätze im Jahr der Anschaffung oder Herstellung und in den folgenden drei Jahren Sonderabschreibungen bis zu jährlich 5 Prozent der Bemessungsgrundlage neben der Absetzung für Abnutzung nach § 7 Absatz 4 in Anspruch genommen werden. ²Im Fall der Anschaffung ist eine Wohnung neu, wenn sie bis zum Ende des Jahres der Fertigstellung angeschafft wird. ³In diesem Fall können die Sonderabschreibungen nach Satz 1 nur vom Anschaffenden in Anspruch genommen werden. ⁴Bei der Anwendung des Satzes 1 sind den Mitgliedstaaten der Europäischen Union Staaten gleichgestellt, die auf Grund vertraglicher Verpflichtung Amtshilfe entsprechend dem EU-Amtshilfegesetz in einem Umfang leisten, der für die Überprüfung der Voraussetzungen dieser Vorschrift erforderlich ist.

(2) ¹Die Sonderabs

1.

durch Bau … r dem 1. Januar 2022 oder
nach dem … antrags oder einer in diesem
Zeitraum … en hergestellt werden, die die
Vorausse … erzu gehören auch die zu
einer Wo

2.

Wohnungen, … dem 1. Januar 2027 gestellten
Bauantrag … ellt werden, in einem Gebäude
liegen, da … e erfüllt und dies durch
Qualitätss

(3) Bemessungsgr … schaffungs- oder
Herstellungskosten der nach Absatz 2 begünstigten Wohnung, jedoch

1.

maximal 2 000 Euro je Quadratmeter Wohnfläche für Wohnungen im Sinne des Absatzes 2 Satz 2 Nummer 1 und

2.

maximal 2 500 Euro je Quadratmeter Wohnfläche für Wohnungen im Sinne des Absatzes 2 Satz 2 Nummer 2.

[…]

Express this
and all the other laws

Lots of it.
Changes all the time.

Subject Matter Expert

Example of the language needed to express such requirements (and similar ones in the domain):

Currencies
Dates
Percentages

Arithmetics
Comparisons
Conditionals
(+wa
Round
Limitir

Summations

Tempo
Year/M

Data/Lookup Tables

Versioning (each year things change)

Testing

With this

Less of it.
Much more stable.

Domain Expert
Software Folks

INTERMISSION

Domain-Driven
DESIGN

How is this related?

Domain-Driven DESIGN

Definition (Wikipedia)

- place the primary focus on the core domain and domain logic; ✓
- base complex designs on a model of the domain (UL);
- initiate a creative collaboration between technical and domain experts to iteratively refine a conceptual model that addresses particular domain problems.

Ubiquituous Language

PLUS (me)

- reify the conceptual model into a DSL that allows the domain experts to directly express subject matter in an executable and testable way.

More Wikipedia: Critics of DDD argue that developers must typically implement a great deal of isolation and encapsulation to maintain the model as a pure and helpful construct.

Working with DSLs is a bit like DDD++ and I am surprised not more DDDers care.

Critics?

I think this isolation is a massive benefit.

Why would you WANT to do that?

Subject Matter Experts are empowered – no longer 2nd class "behind" devs.

Devs can focus on technical concerns, don't have to unde

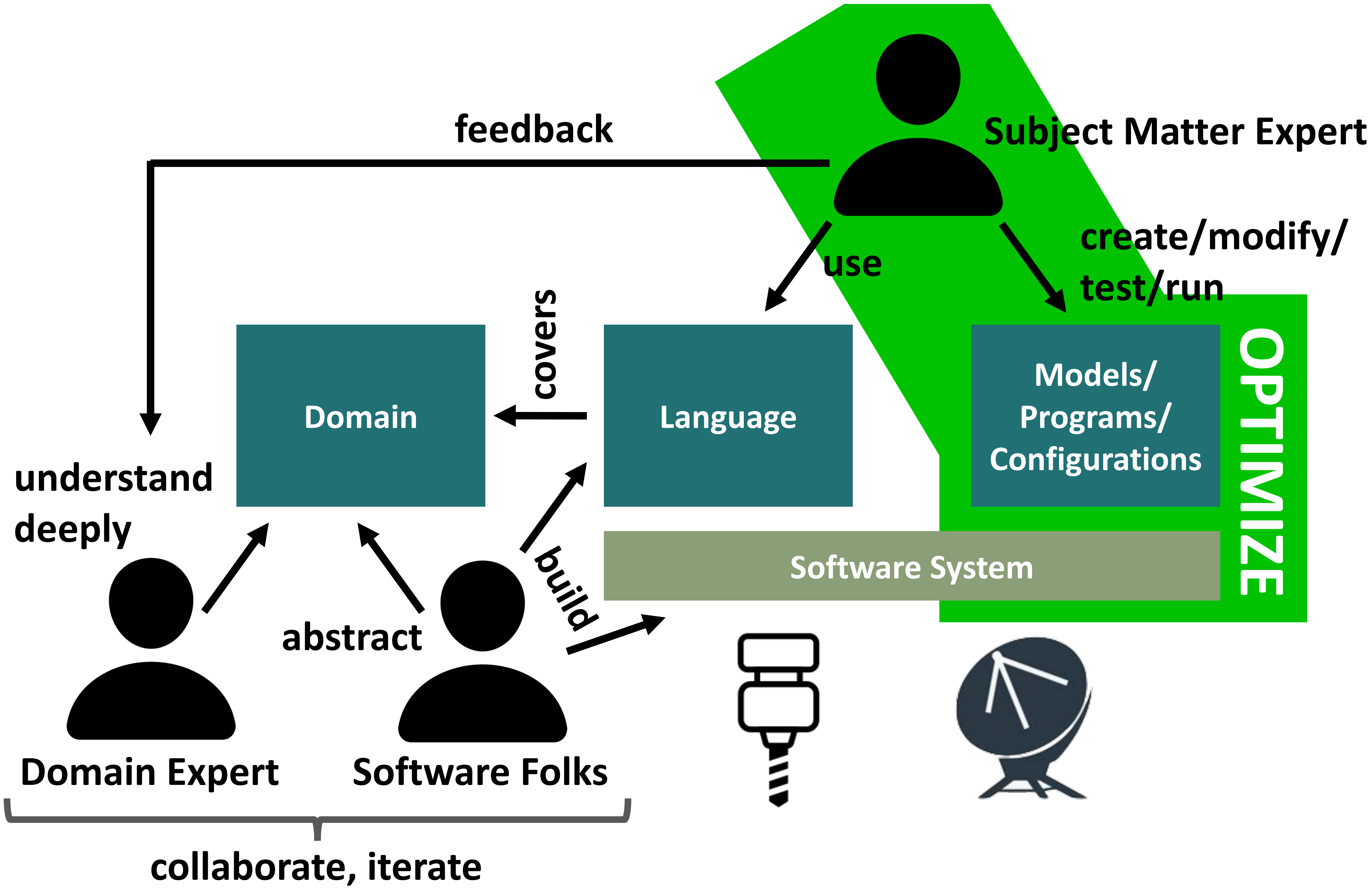
Subject matter is portable, the legacy problem is much re

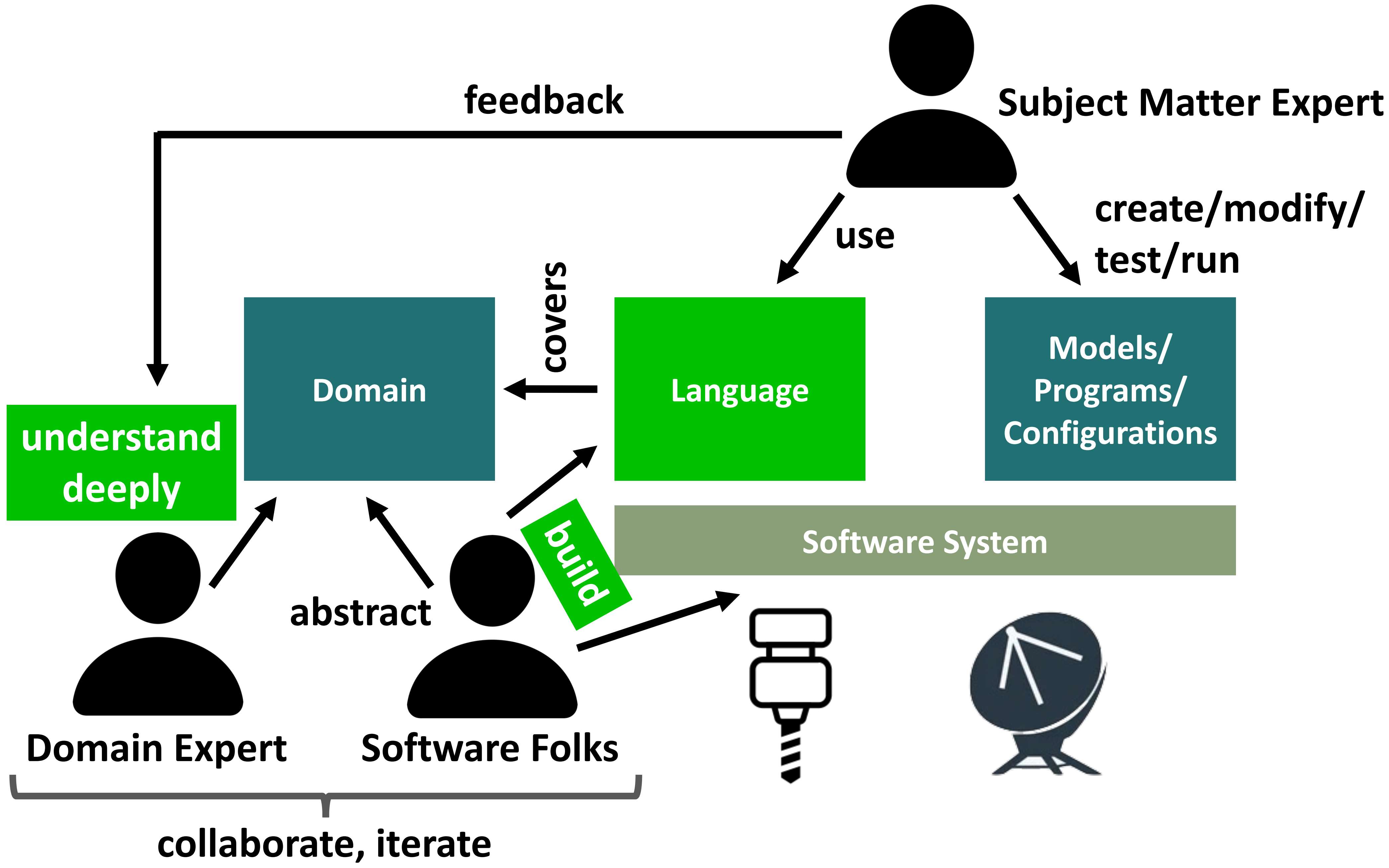
Collaboration between SMEs and devs better because fo

Done right, the overall subject matter development proc

https://www.linkedin.com/pulse/relationship-between-domain-driven-design-languages-markus-voelter

AND NOW BACK TO
OUR REGULARLY
SCHEDULED
PROGRAMMING


feedback
Subject Matter Expert
use
create/modify/
test/run
covers
Domain
Language
Models/
Programs/
Configurations
OPTIMIZE
understand
deeply
Software System
build
abstract
Domain Expert
Software Folks
collaborate, iterate


feedback
Subject Matter Expert
use
create/modify/
test/run
covers
Domain
Language
Models/
Programs/
Configurations
understand
deeply
build
Software System
abstract
Domain Expert
Software Folks
collaborate, iterate

A whole bunch of ingredients

Understand the domain

Design the Language

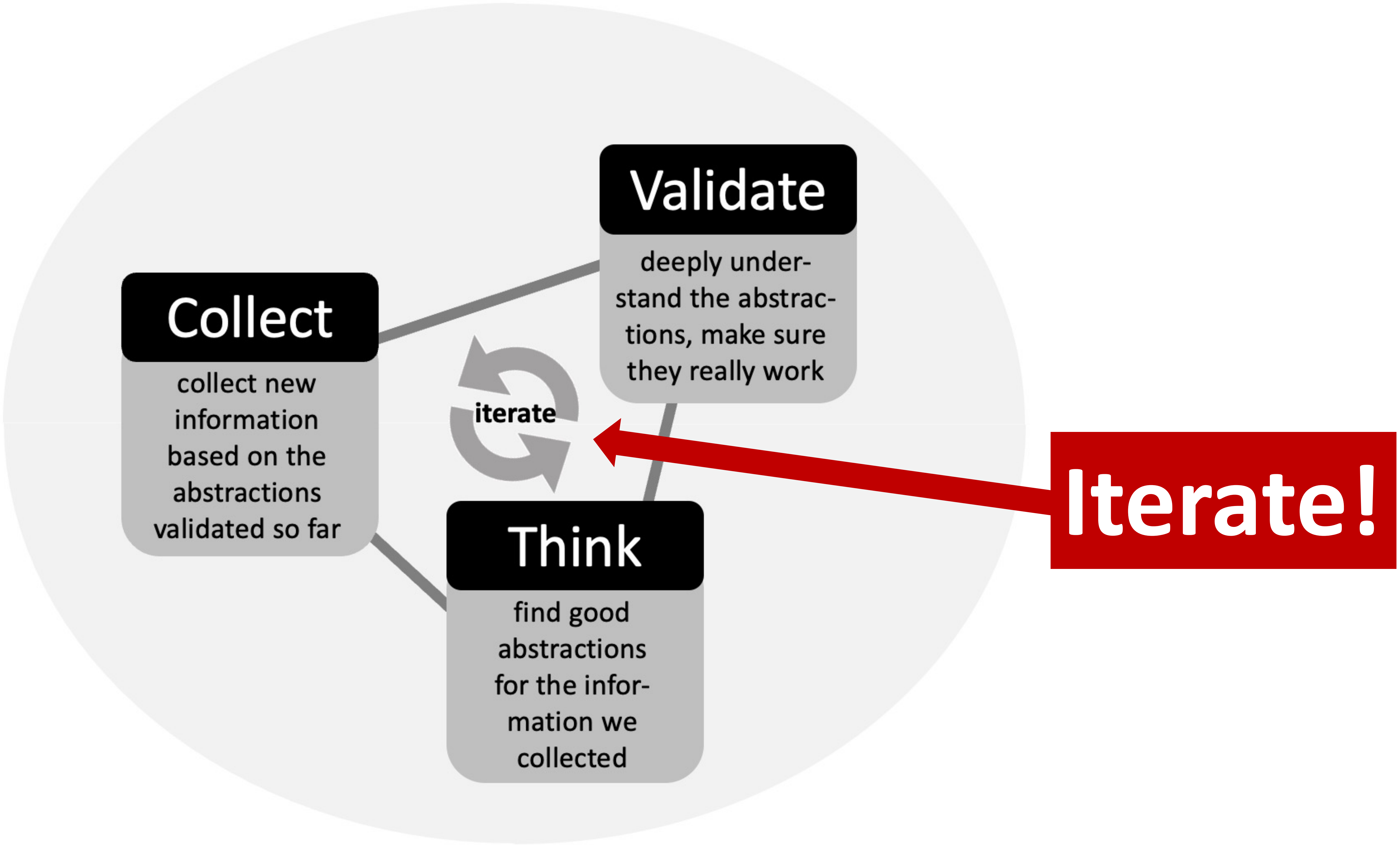
Implement language and tools

Architect the software system

Implement runtimes & generators

Introduce to organisation

Understand the domain


Collect
collect new information based on the abstractions validated so far
Validate
deeply understand the abstractions, make sure they really work
iterate
Think
find good abstractions for the information we collected
Iterate!

COLLECT

Survey the Land

Bounds of the domain.

Say what's out, and why.

Connections to the surroundings.

Similar to context diagrams in software architecture.

People over stuff

People

Documents

Code

{

Outdated
Imprecise
Loveless
Status Quo

People over stuff

People

Documents

$$\text{Code} \begin{cases} \text{Status Quo} \\ \text{Hidden Domain Semantics} \end{cases}$$

Hidden Languages

People
Documents
Code

Hidden Languages

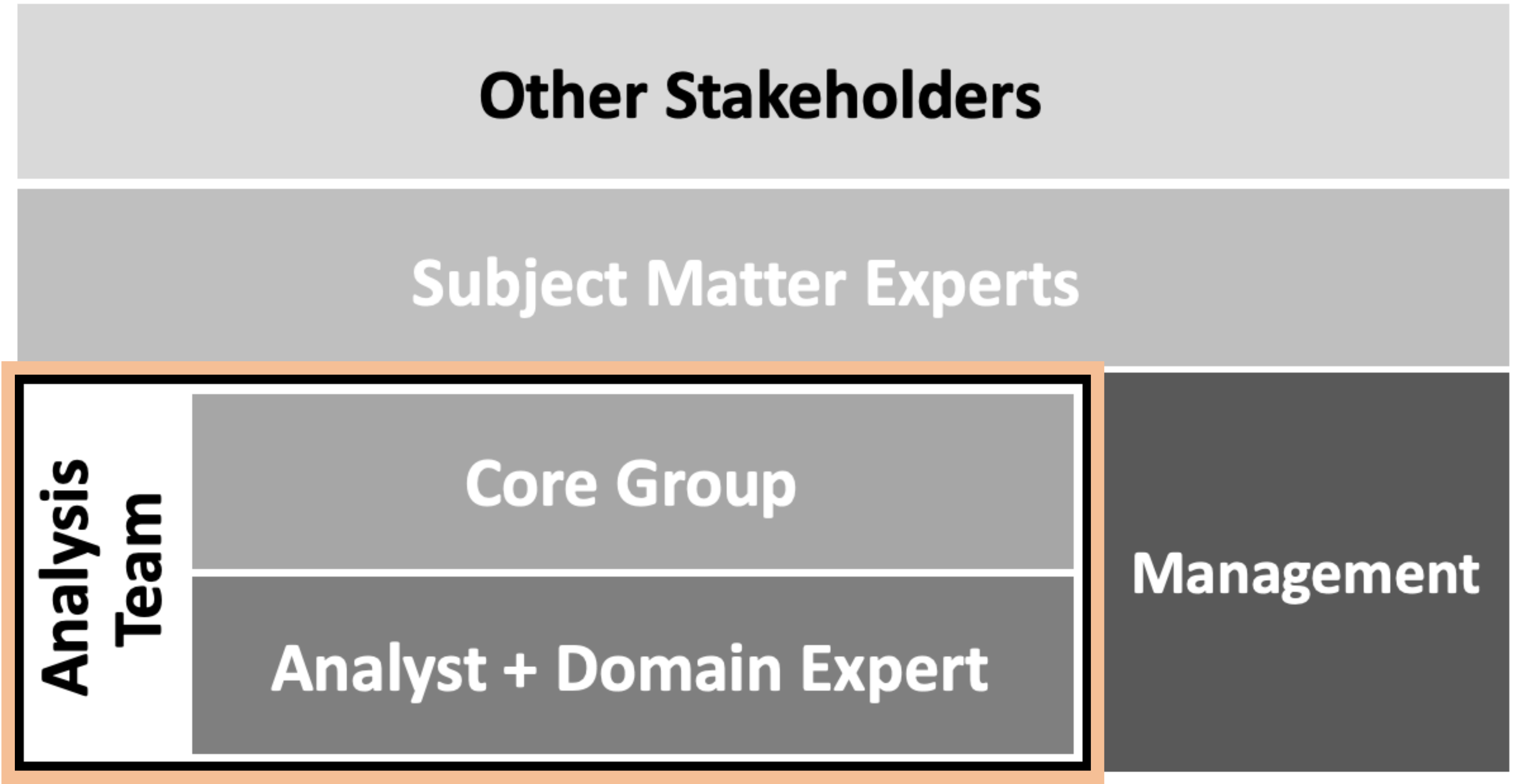
Word templates
XML Schemas
Excel Sheets

People over Stuff

People

3 .. 7

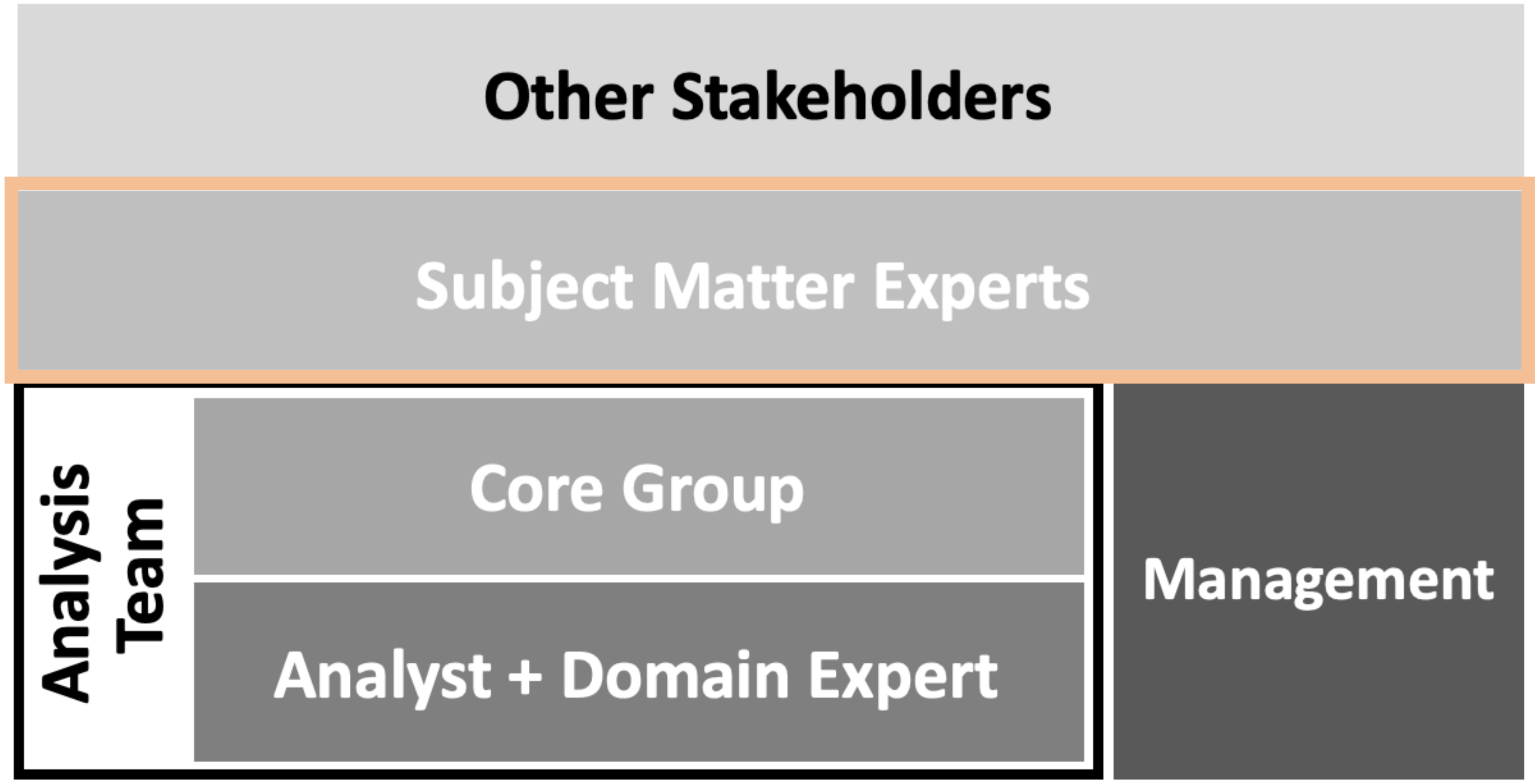
Participate regularly and deeply understand the results



People over Stuff

People

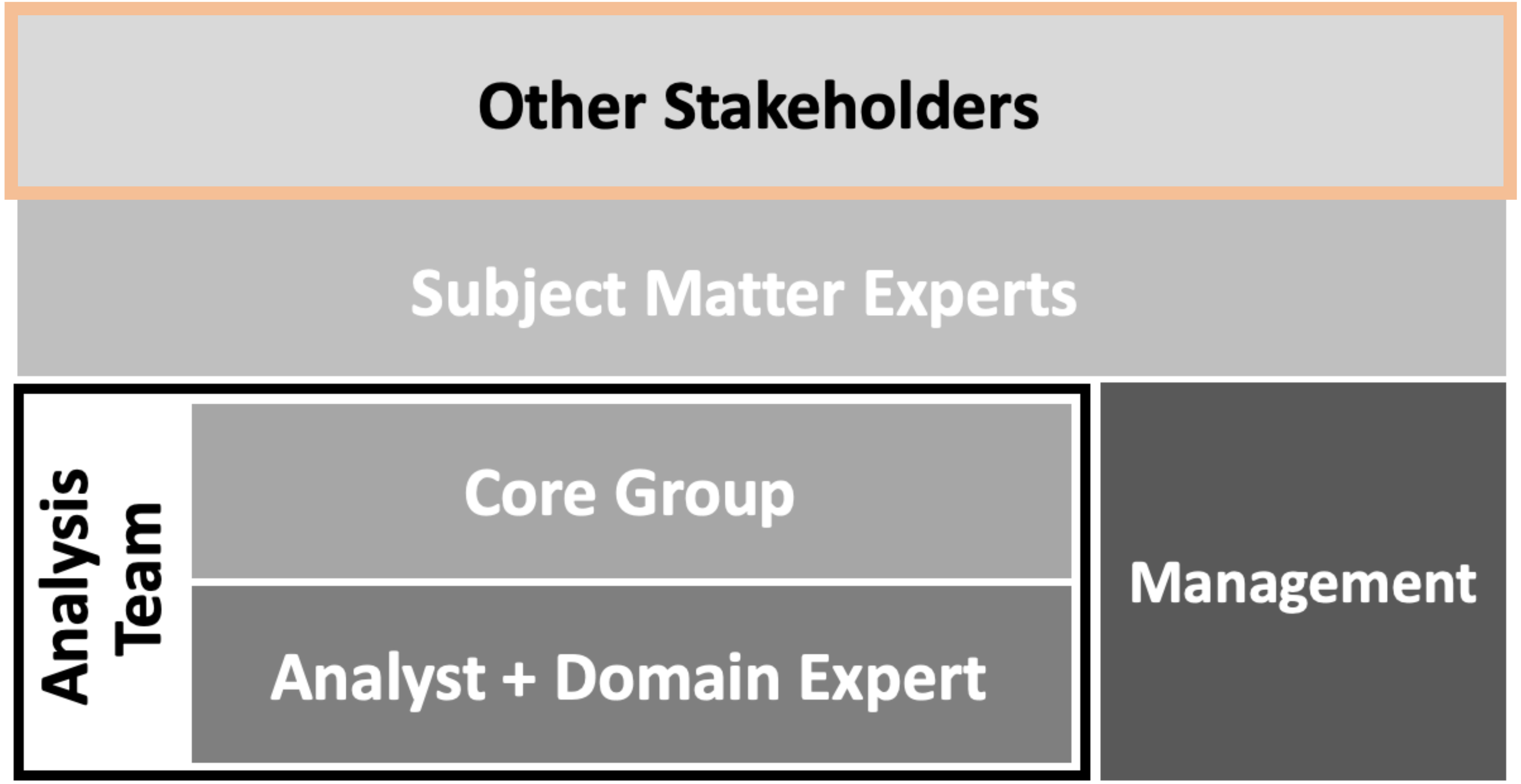
Validate results, contribute feedback and special topics



People over Stuff

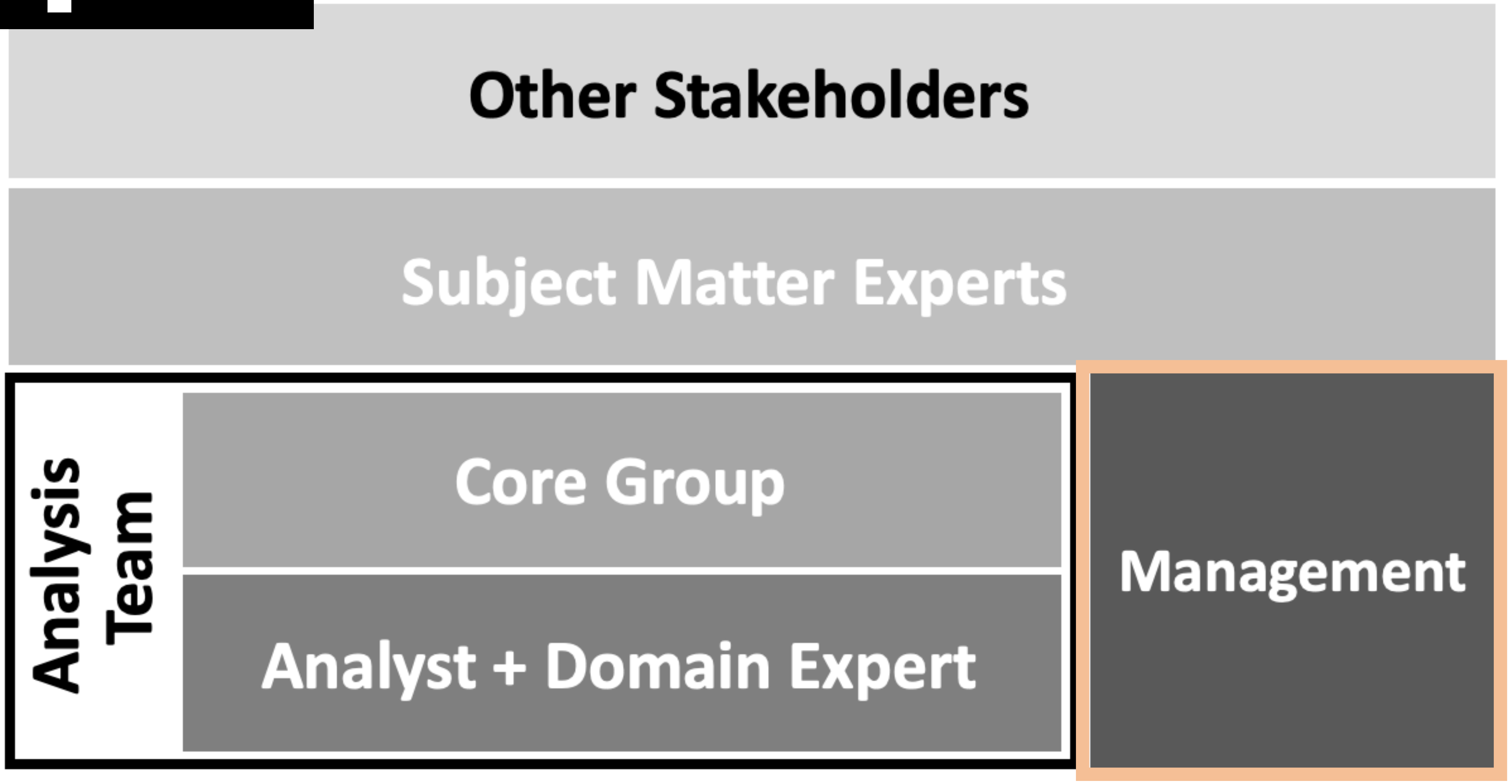
People

Less regular feedback



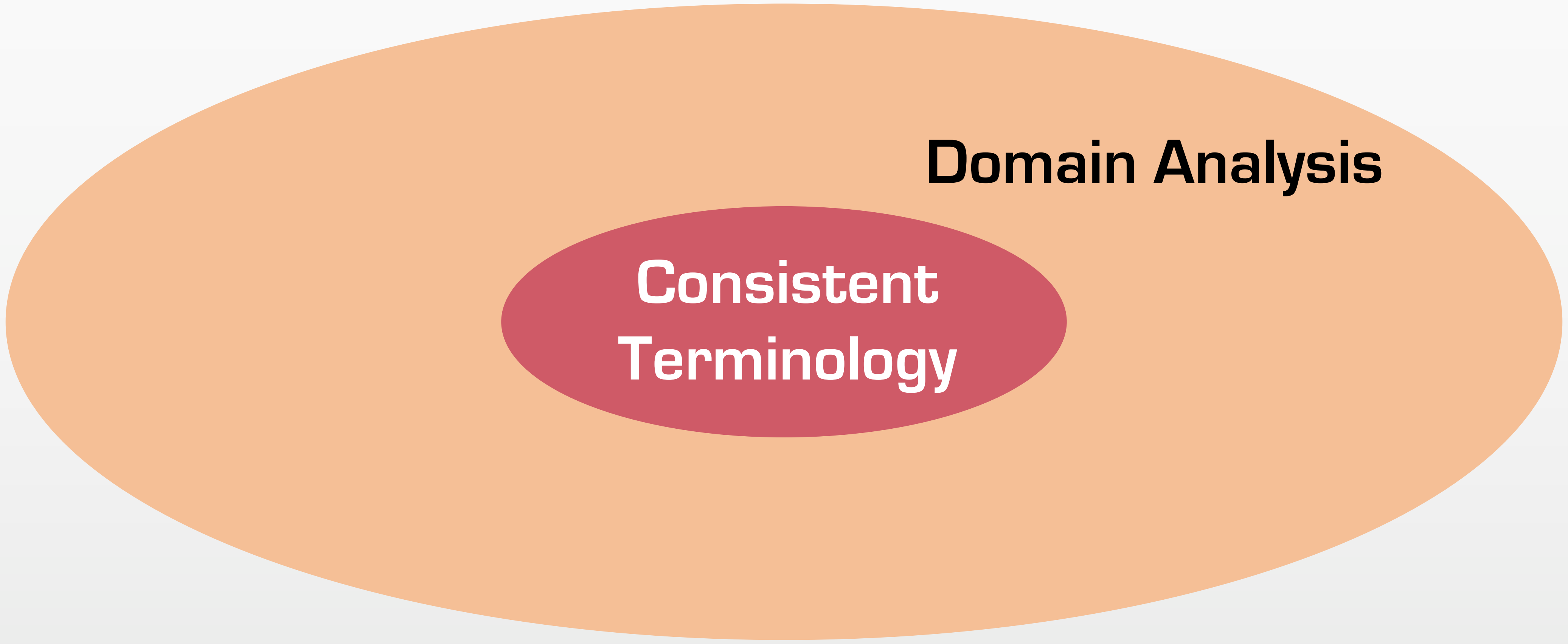
People over Stuff

People



Scope, priorisation, decisions that affect business

Consistent Terminology



Domain analysis is much more than a glossary; but without agreed terminology, a domain analysis cannot succeed.

Workshops

Primary means of collecting information and feedback.

Workshops

The basics: Moderation

Ensure regular **breaks**

Allow **everyone** to speak

Have an **agenda** and **stick** to the topic(s) at hand

Steer towards **conclusions** (and not just info exchange)

Capture **results** and open issues

Stop straw men & unrealistic, simplified statements

Shut down ‘sabotage‘ (ego, politics)

Workshops

Workshops

Steering the analysis

Distinguish good and bad **quarrels**

Identify **rabbit holes**, and make sure you come up again.

Build a **mental model** and

- detect holes, inconsistencies
- verify the "right" abstraction level?
- give (counter)**examples** of the mental model
- encourage others to give examples.

Make sure everybody understands **agreements**.

Workshops

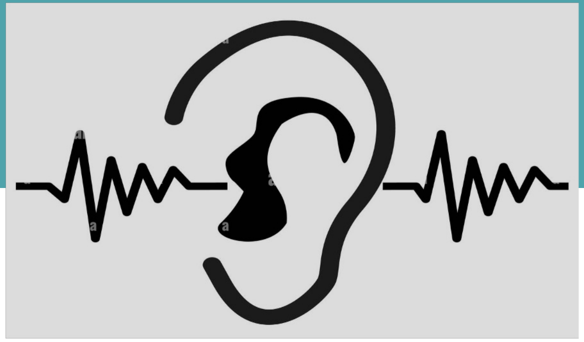
Workshops

3 hours per session

4 sessions per week

Keep the atmosphere friendly and professional. Bring cookies.

Active Listening



Re-explain in your own words what you have understood

State explicitly what (you think) a speaker's words don't mean

Rephrase what was told in terms of the abstractions we have found so far

Point out if a speaker:

... makes **implicit assumptions** without saying those

... is **imprecise** (in terms of content or terminology)

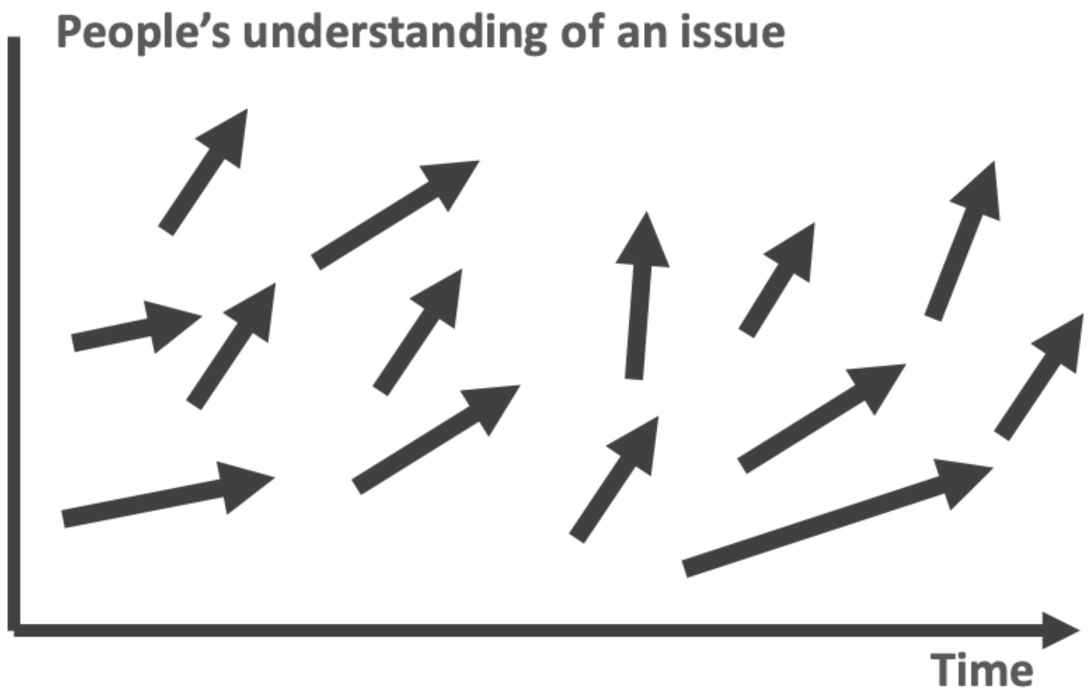
... **contradicts** previous agreements (to reconsider either one)

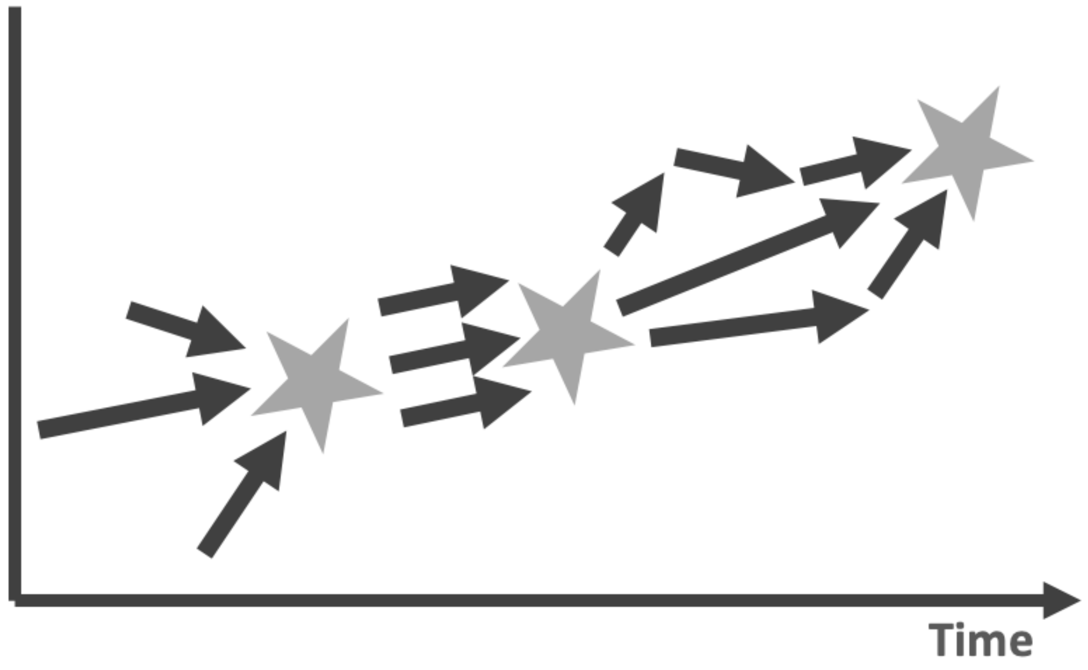
... **mixes** different questions or aspects of the problem

Beware of appearing arrogant.
Acknowledge before criticism.

Beware of overwhelming the team.
You are probably the most meta-fluent.

Consistency vs. Change





At any point in time, the analysis team has a **joint, consistent understanding** of the domain. This will **evolve** over time.

Dealing with Uncertainty

Put it into a box so it doesn't spread uncontrollably.

- What is the **precise problem** about which there is uncertainty?
- What are the **alternative solutions** to that problem, plus tradeoffs and examples?
- What **adjacent** or **related** questions are **certain**; what can we agree on?

the box

Capture Results

Similar to architecture decision records.

I suggest to use an issue tracker, because issues can be identified, commented, prioritised, searched. BAN EMAIL!

Decisions plus Rationale – what have we decided, and why.

Scope and (counter)examples.

Open Questions – what do we want to try to understand next

Disagreements – where can we currently not agree

Keep it pragmatic, or it won‘t happen at all.

This is a team‘s working tool. Not for public consumption.

See also Domain Spec and Domain Impl.

THINK

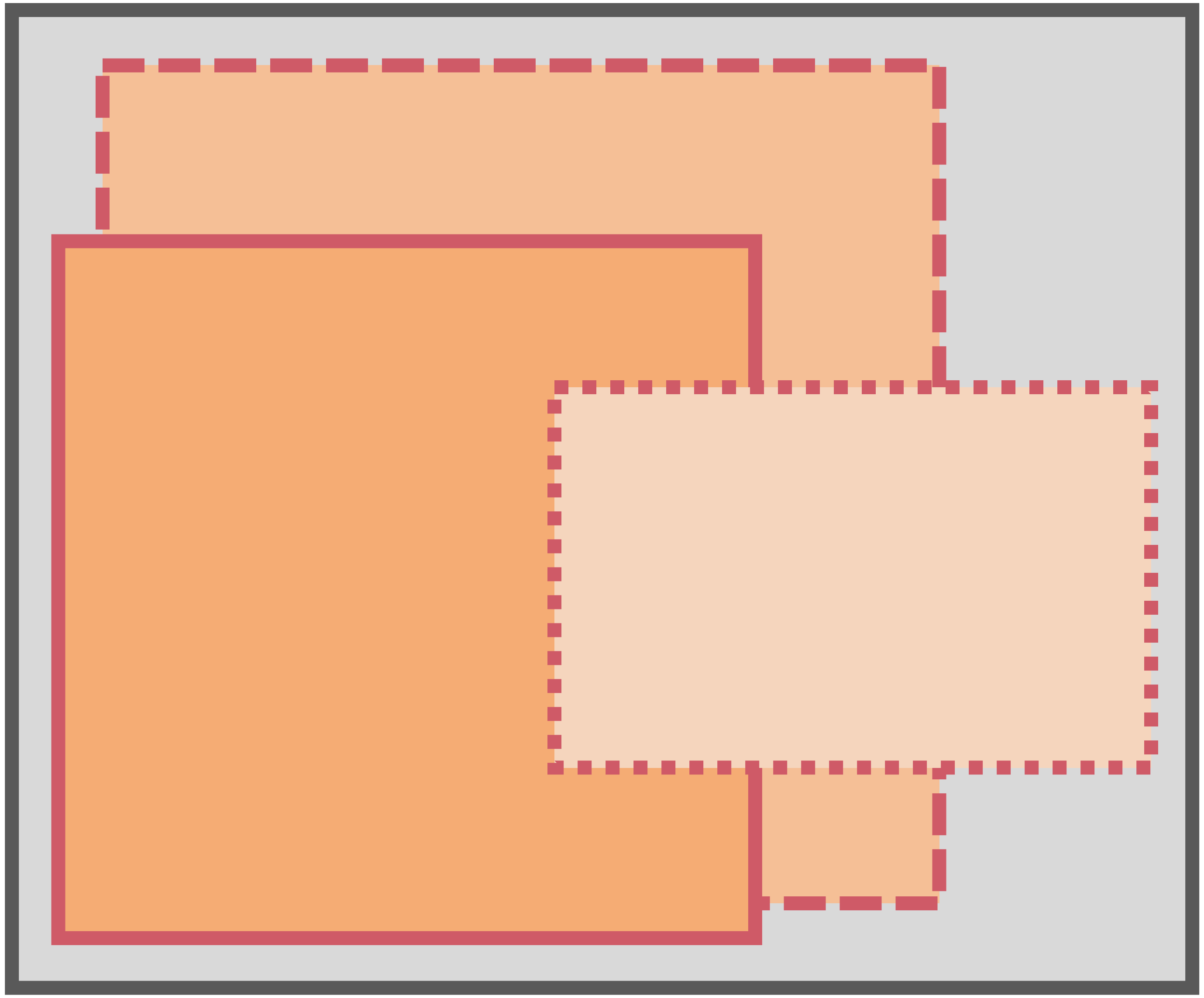
Take time to think!

Completely full calendars prevent deep thinking and conceptual work.

A trivial statement, but a problem in
In many organisations nonetheless

Often best done in pairs.
Eg. Analyst + Domain Expert

Bounds of the Domain

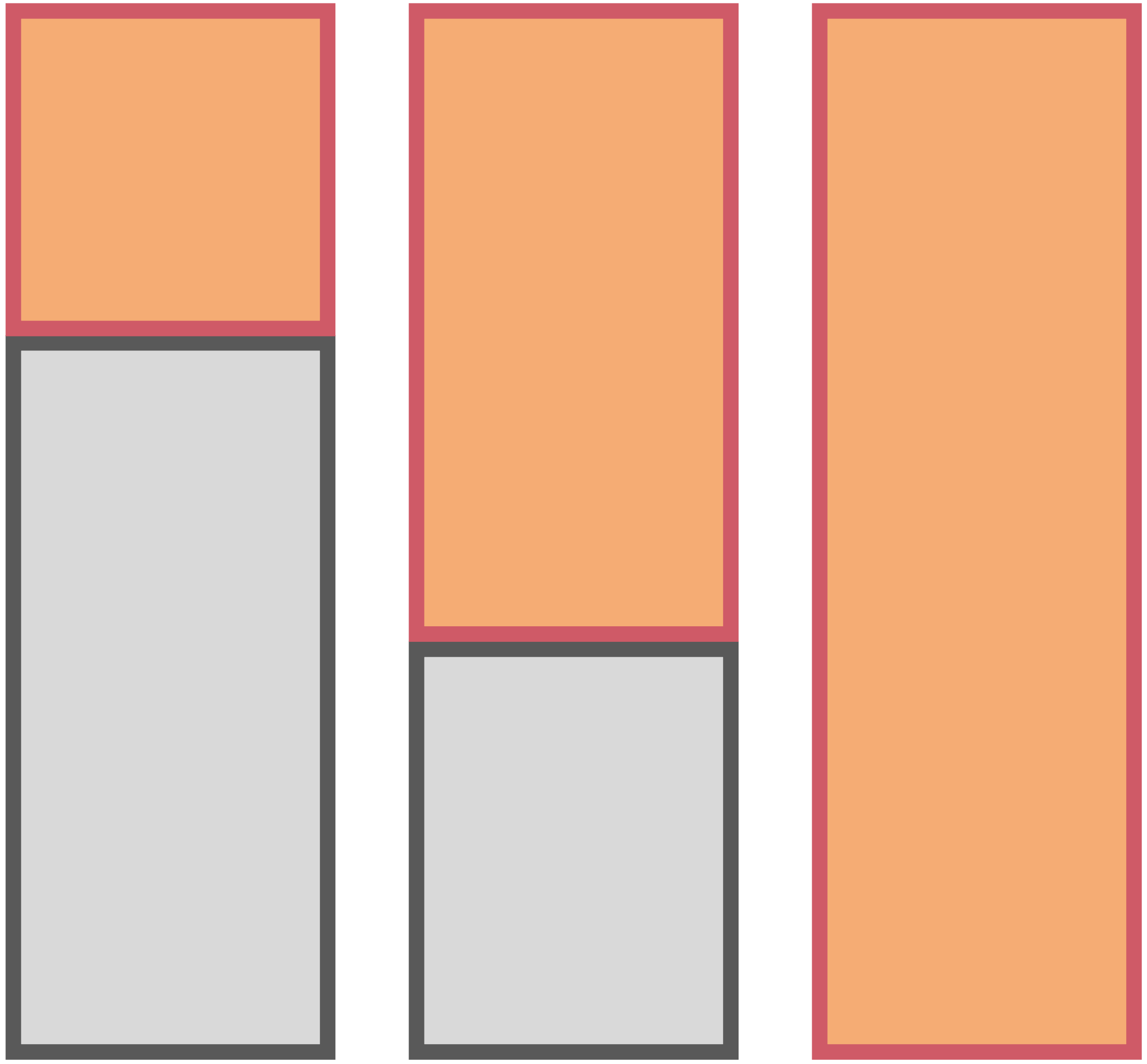


Which parts of the domain should we include?

Will this destroy abstractions?

Foundational Question.

Depth of the solution

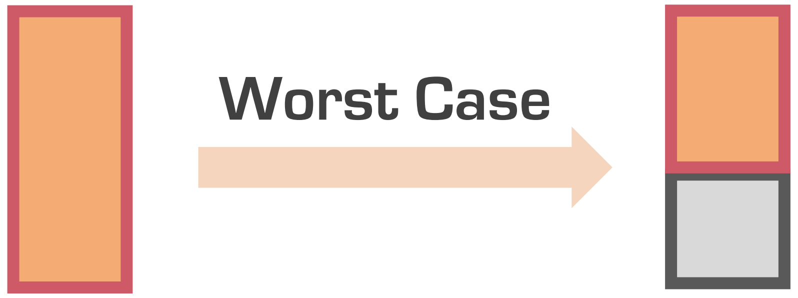


Should we only rething/-write the „application layer“ and keep the backend infrastructure the same?

Potentially lots of Complexity.

Old system pollutes new abstractions

Big mapping effort – Semantics!



Identify, Question and Remove Cruft

Source of potentially unjustifiable complexity.

Historic accidents | Pragmatic shortcuts | Special solutions for (former) customers | Hacks to get things done or make things fast | Features that are no longer necessary | Changes in business strategy and prioritites

Identify, Question and Remove Cruft

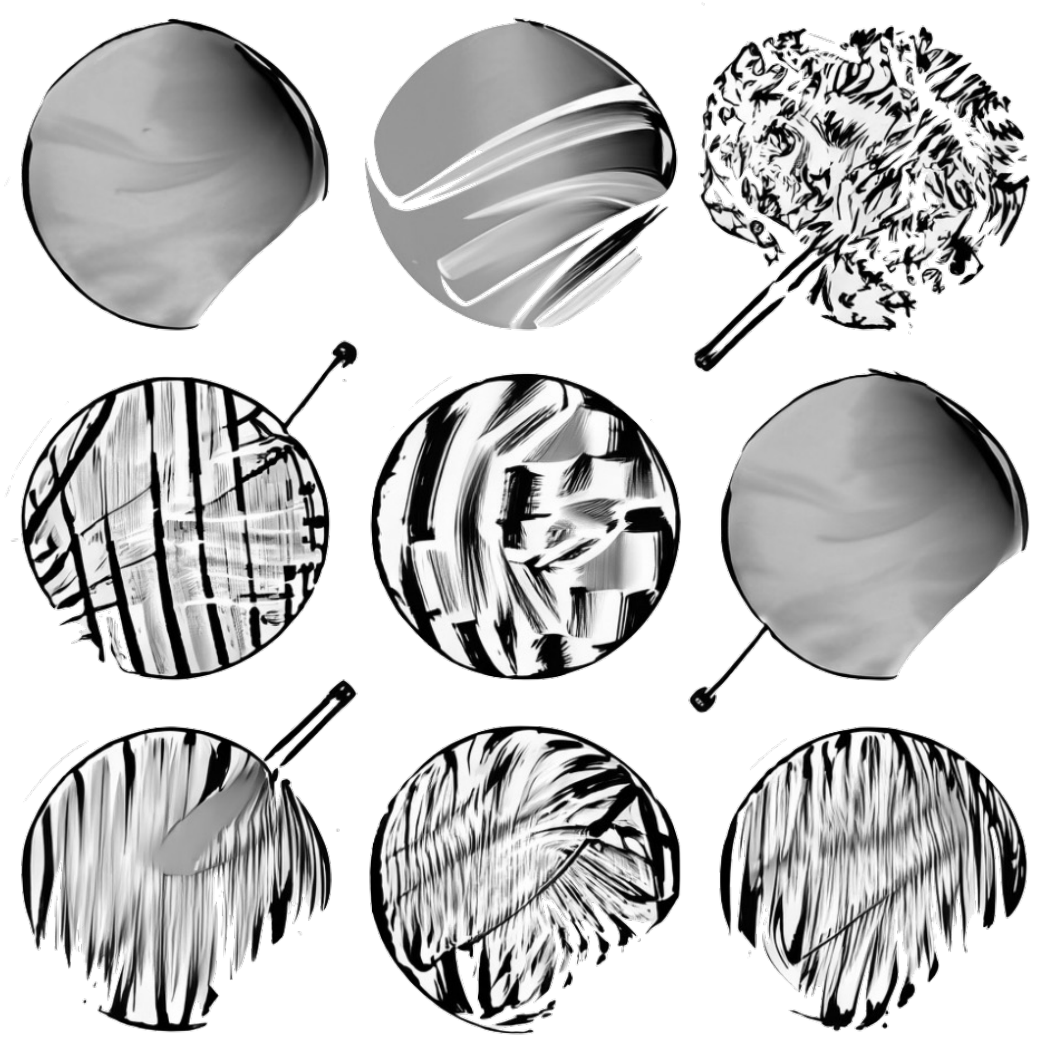
Source of potentially unjustifiable complexity.

Not always easy to **identify** (`if customerSpecialFlag then`)

Even harder to **decide to kill** it because nobody remembers reason why its there (but there must be one…). Management.

Illustrate the **price** you pay for keeping.

Abstraction



Cover the **whole** domain

Semantics for purpose of the tool

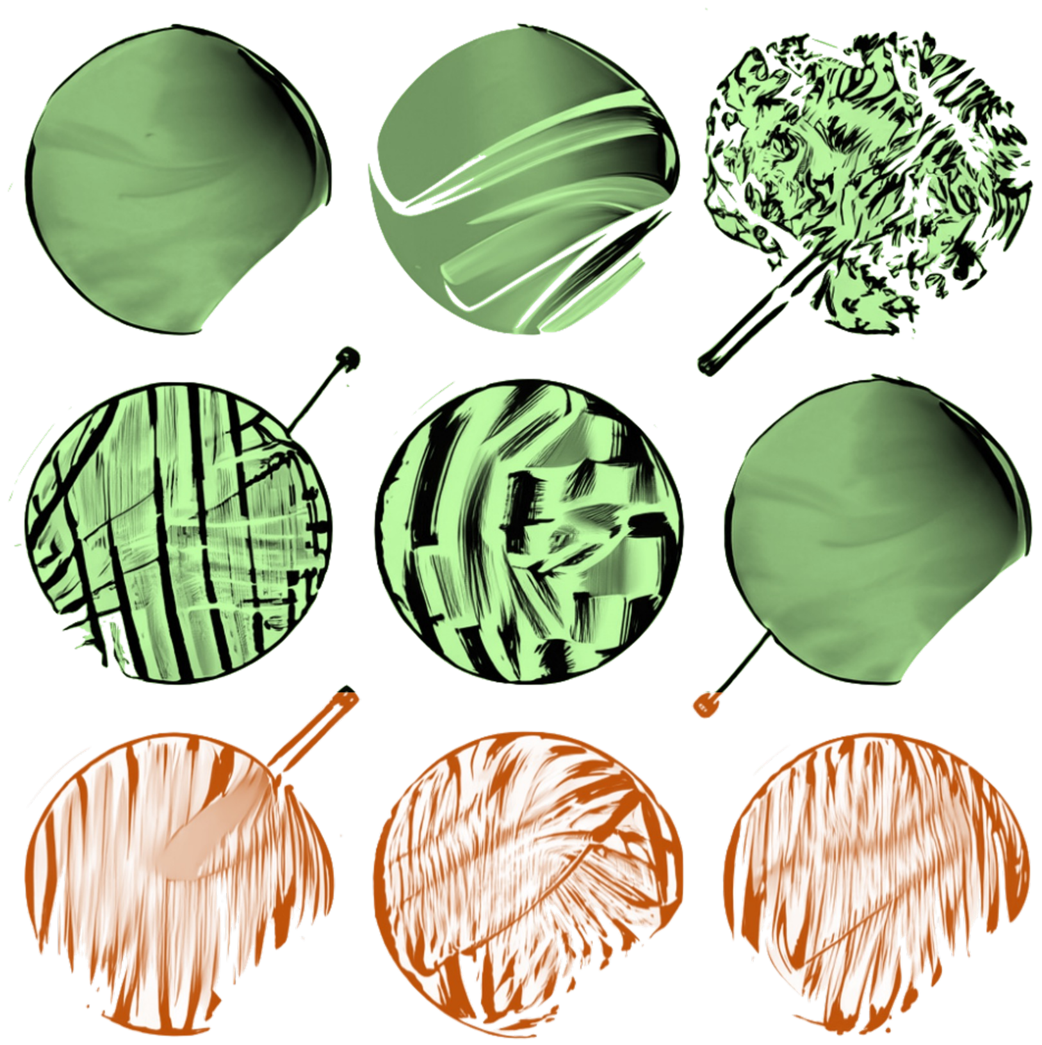
Expressive Power vs. Learnability

Genericity vs. Specificity

Based on experience

+ a whole chapter of details in the book

Abstractions for Testing

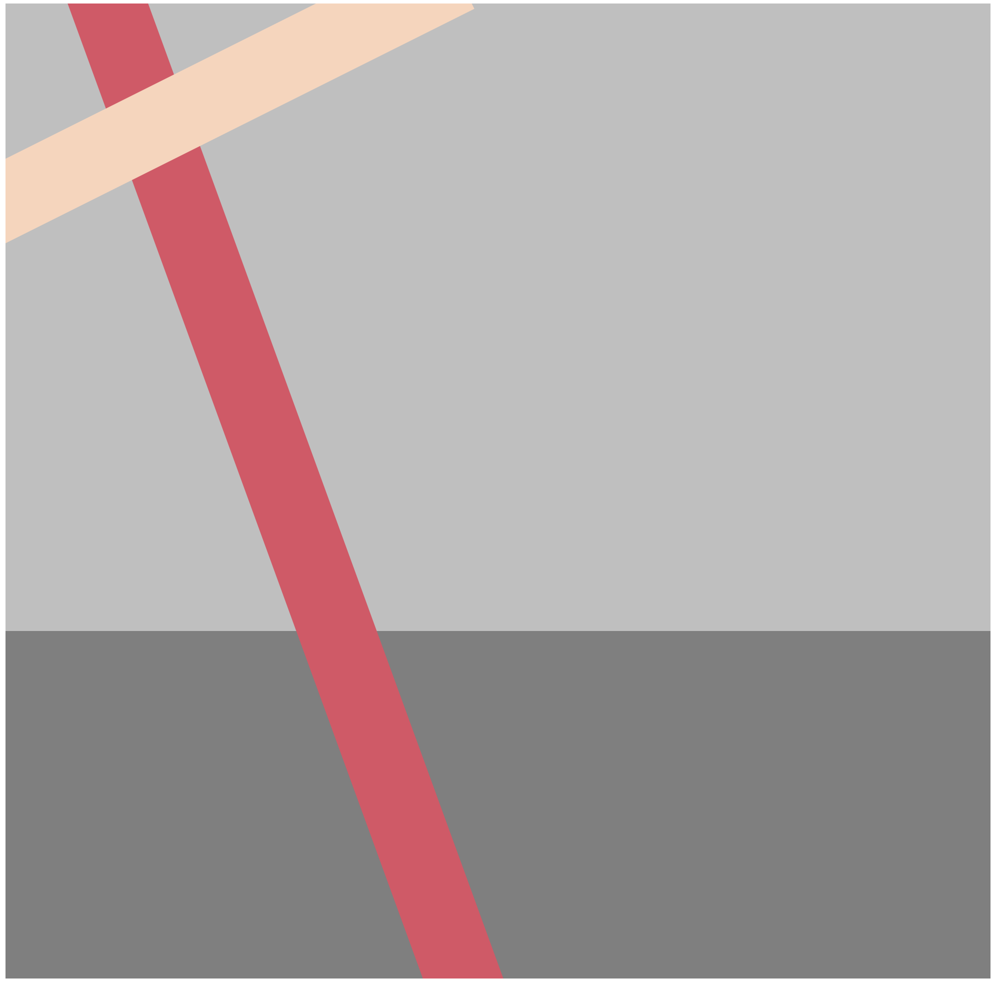


Cover the **whole** domain
Semantics for purpose of the tool

Expressive Power vs. Learnability
Genericity vs. Specificity

Based on experience
+ a whole chapter of details in the book

Platforms and Crosscuts



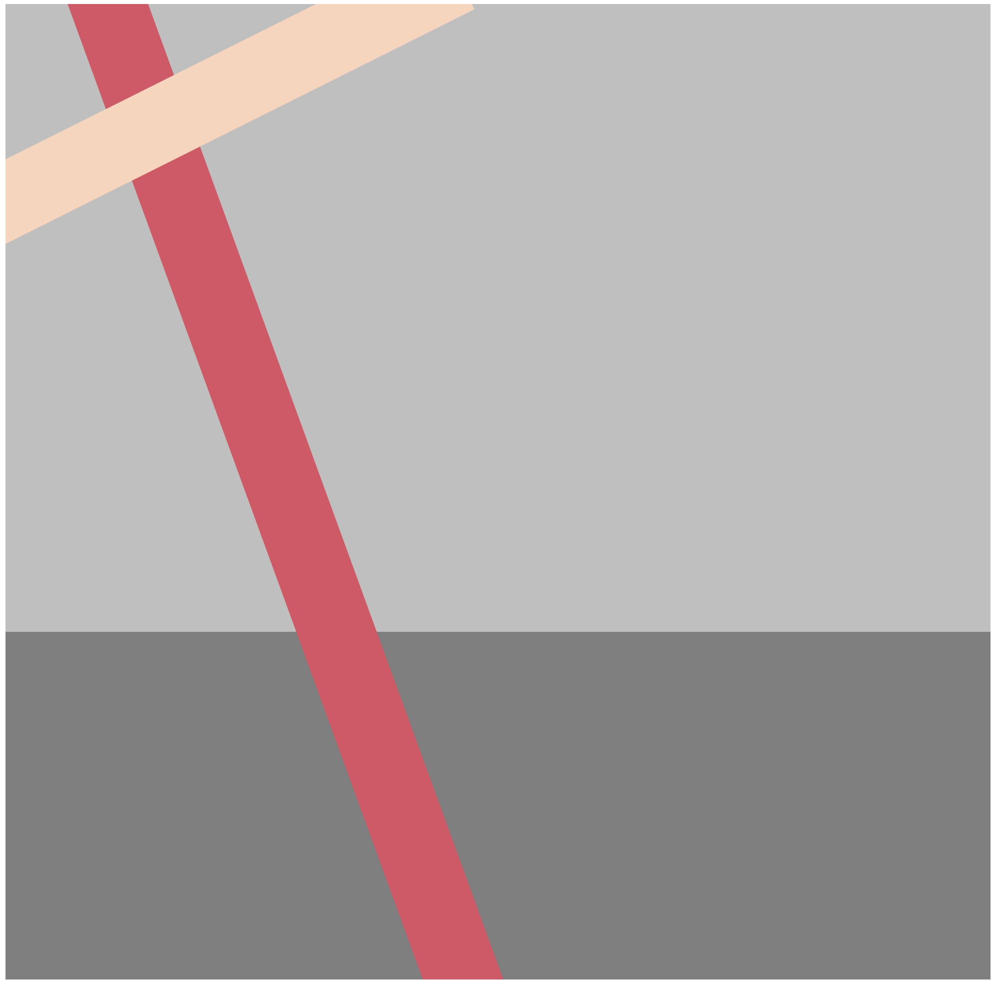
The **platform** contains things that are foundational, that are the same every-where, and therefore do not have to be described by the language your are trying to build.

e.g. persistence, transport, UI

Crosscuts are things that should be done **consistently** throughout the system and affect many parts of the language.

e.g. dealing with time, versioning

Platforms and Crosscuts



Platform

often a relatively natural outcome of the domain analysis.

Crosscuts

often hard to discover.
often where the value lies.

Industry Standards

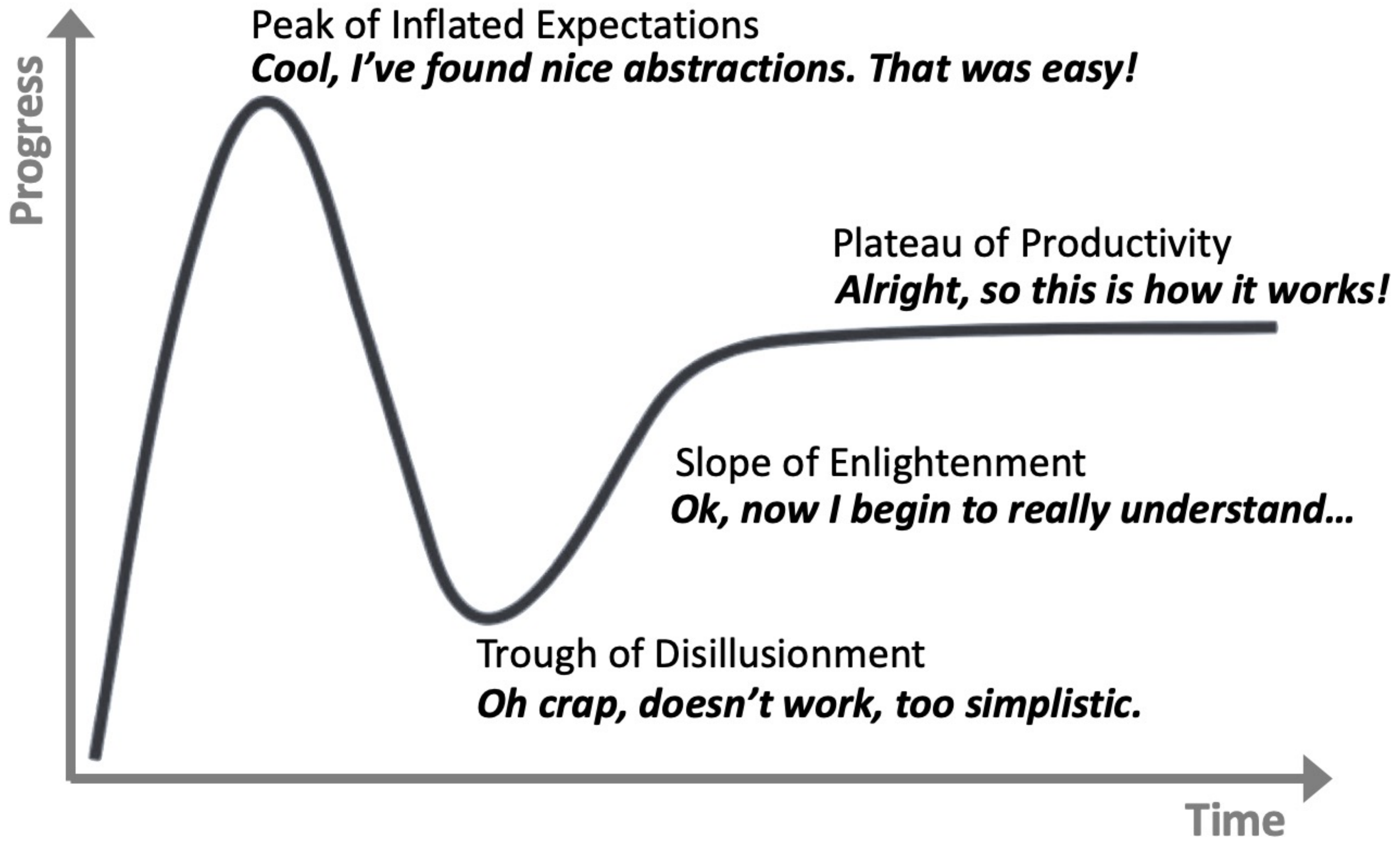
Double-edged sword.

Ups and Downs

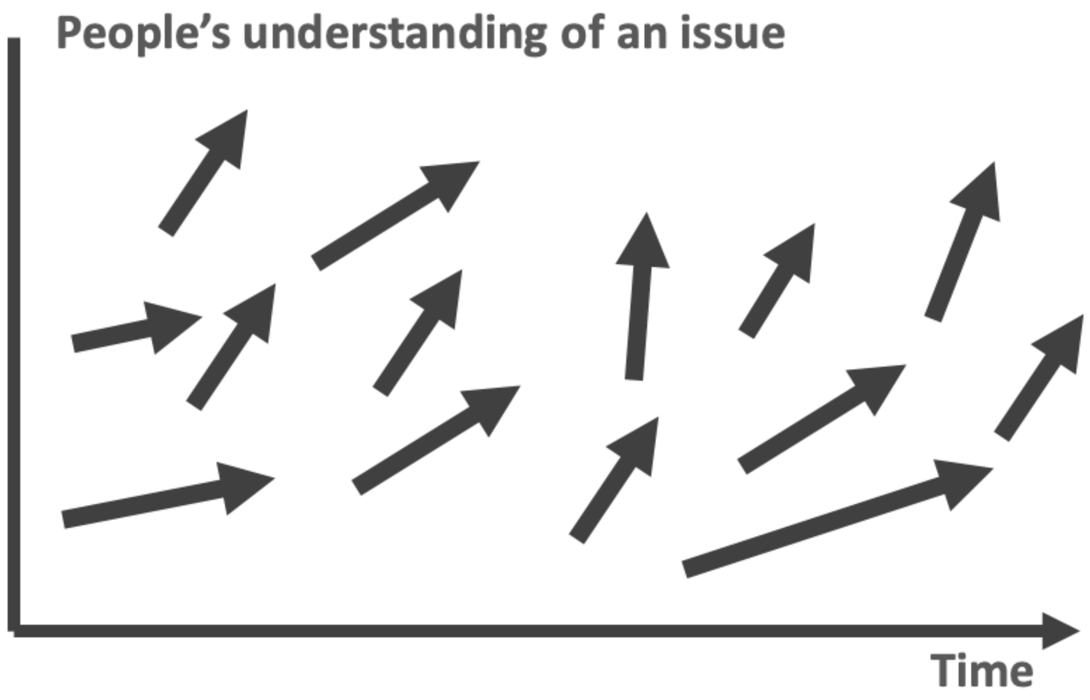
It's structurally similar to the well known hype cylce

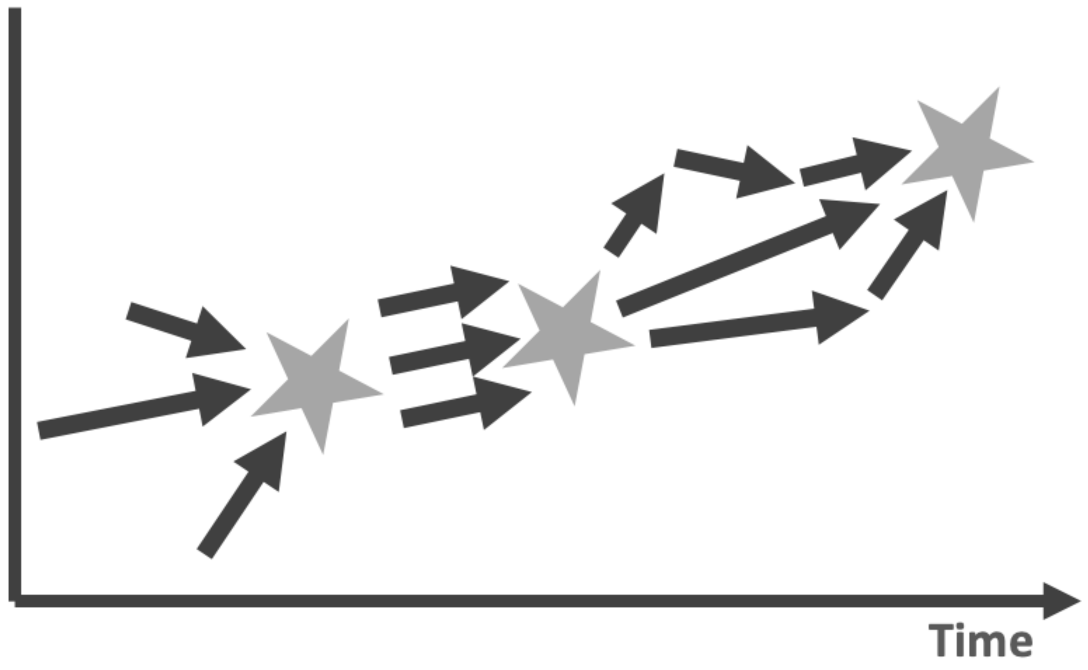
Don't give up too early!

Multiple swings are possible; ok if amplitude becomes less over time.


Progress
Peak of Inflated Expectations
Cool, I've found nice abstractions. That was easy!
Plateau of Productivity
Alright, so this is how it works!
Slope of Enlightenment
Ok, now I begin to really understand...
Trough of Disillusionment
Oh crap, doesn't work, too simplistic.
Time

Spread the Knowledge





Make sure everybody **really** understands the state of the thinking and the abstractions!

Push and pull.

IV

VALIDATE

Domain Specification

Trying to write clearly and understandably helps you think!

A document:

Literature. A book. Explain, illustrate. Read on the subway.

- Keep rationales brief. Point to issue tracker.
- Use informal models / diagrams to illustrate.
- Use lots of examples. Most people learn by example.
- Don't give all the details. Emphasize concepts and the gist.

Not slides. Not issues. Not a tool tutorial.

https://www.linkedin.com/pulse/stop-using-slides-markus-voelter

Papier ist geduldig.

Bubbles don’t crash.

Domain Implementation

You can't validate
"just words"

You need an executable prototype
of the language.

Domain Implementation

You can't validate "just words"

You need an executable prototype of the language.

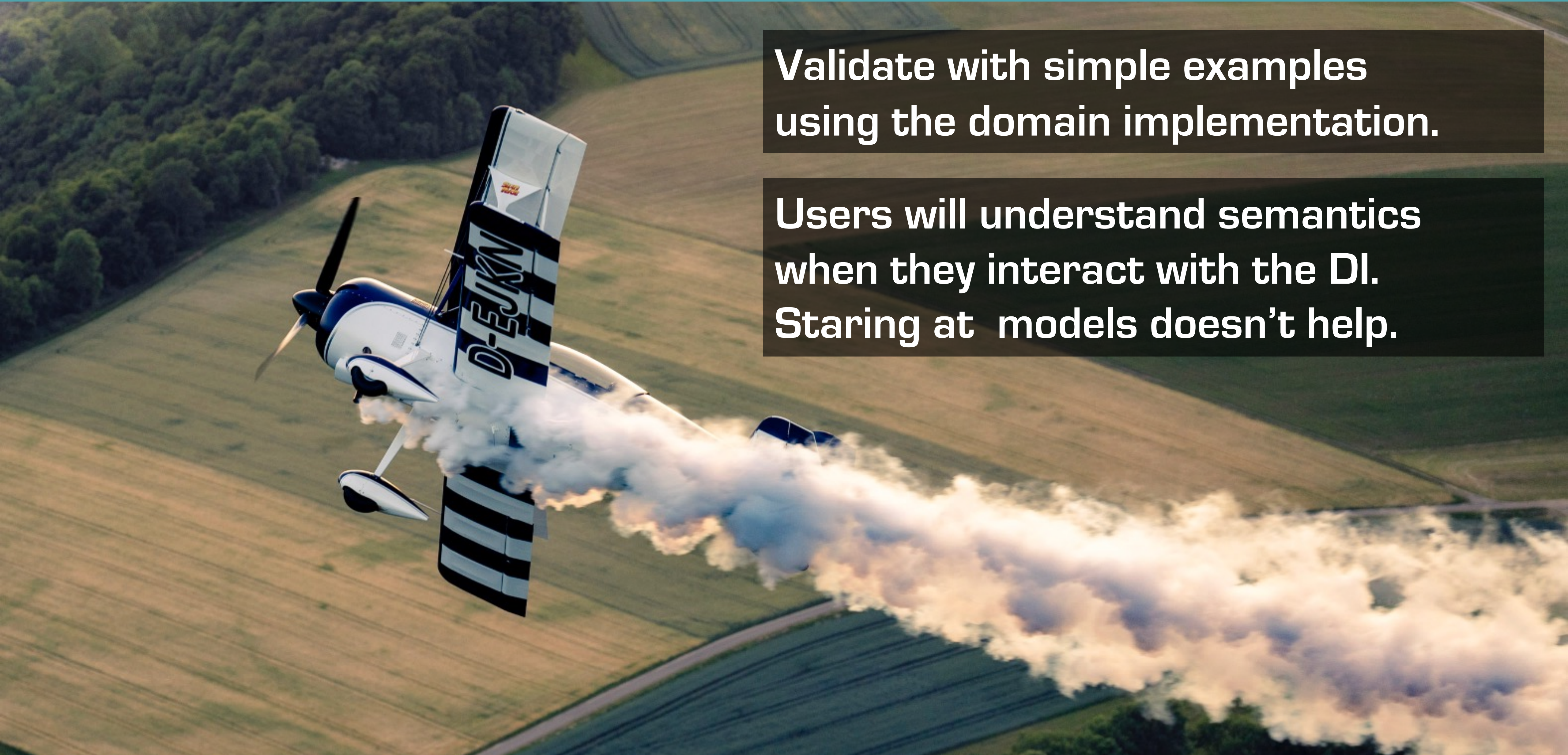
MPS

Formality forces consistency and completeness.

Execution helps with validation.

Must be fast so you can iterate daily.

Let users play!



Validate with simple examples using the domain implementation.

Users will understand semantics when they interact with the DI. Staring at models doesn't help.

Let users play!

Build something real!

Use the DI to implement representative realistic real-world cases

Conceptual Review

Eg., based on Cognitive Dimensions of Notations

Abstraction gradient | Consistency | Diffuseness versus terseness | Error-proneness | Hard mental operations | Hidden dependencies | Role-expressiveness | Viscosity | Premature commitment

Analyse Usage

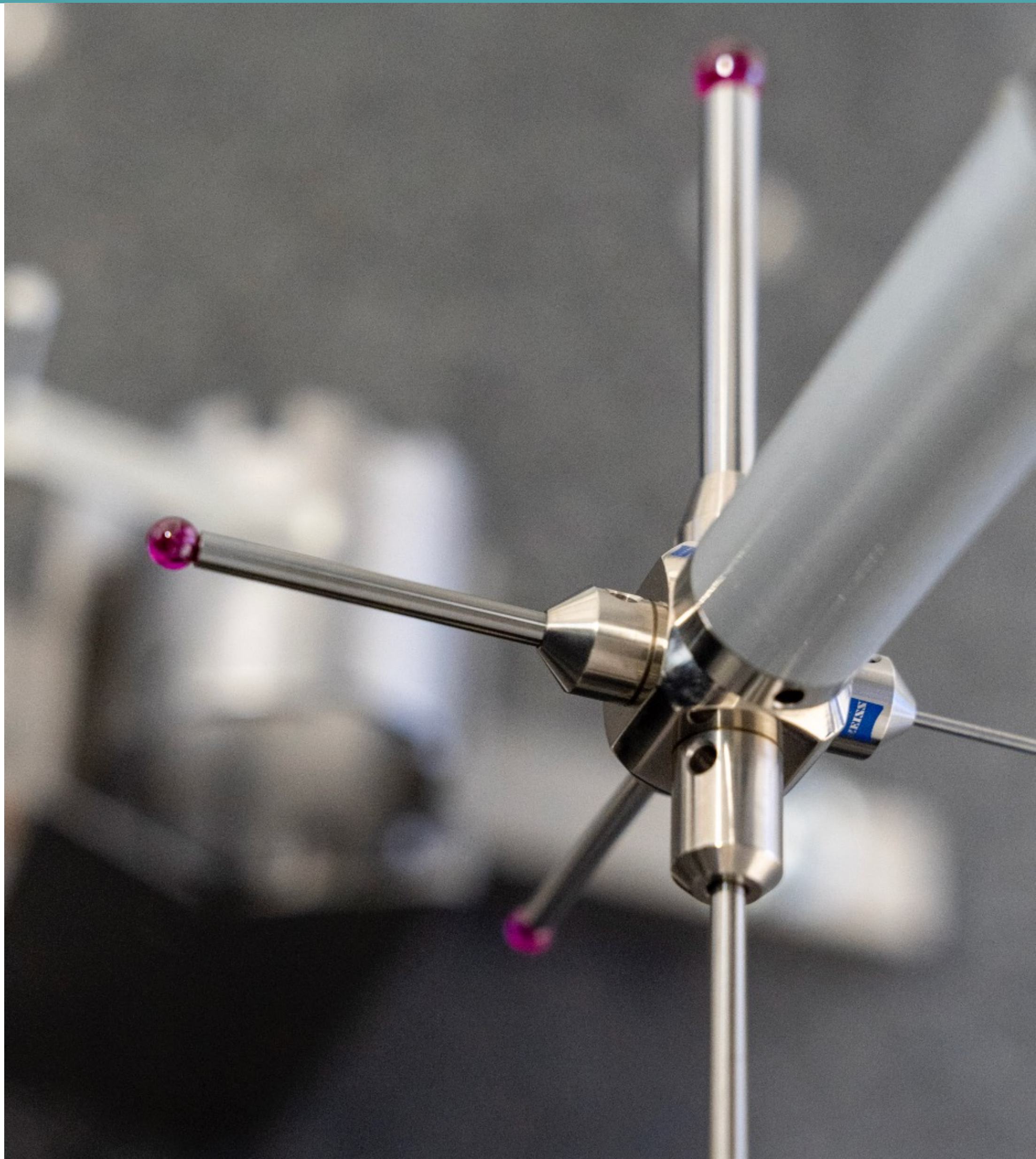
Review how concepts are **used**.

‘**Misuse**’ is relevant input.

Combined use suggest new abstraction

Quantitative data suggests trade-offs.

Potentially automatable.



VI

CROSS-CUTTING

Dealing with Feedback

Ideal Case

- You **receive** feedback that uncovers a conceptual problem
- You **think** it through, then adapt the language accordingly
- In the next workshop you **report** the feedback,
  explain the change you made
  ask for feedback on that change

Be sure to **credit** the person who had the idea or provided the initial feedback.

Small? Fix it right then an there, don’t create an issue.

A bit bigger? Write the issue while the feedback can read and check it.

Dealing with Feedback

Ideal Case

- You receive feedback that uncovers a conceptual problem
- You think it through, then adapt the language accordingly
- In the next workshop you report the feedback,
- explain the change you made
- ask for feedback on that change

Be sure to **credit** the person who had the idea or provided the initial feedback.

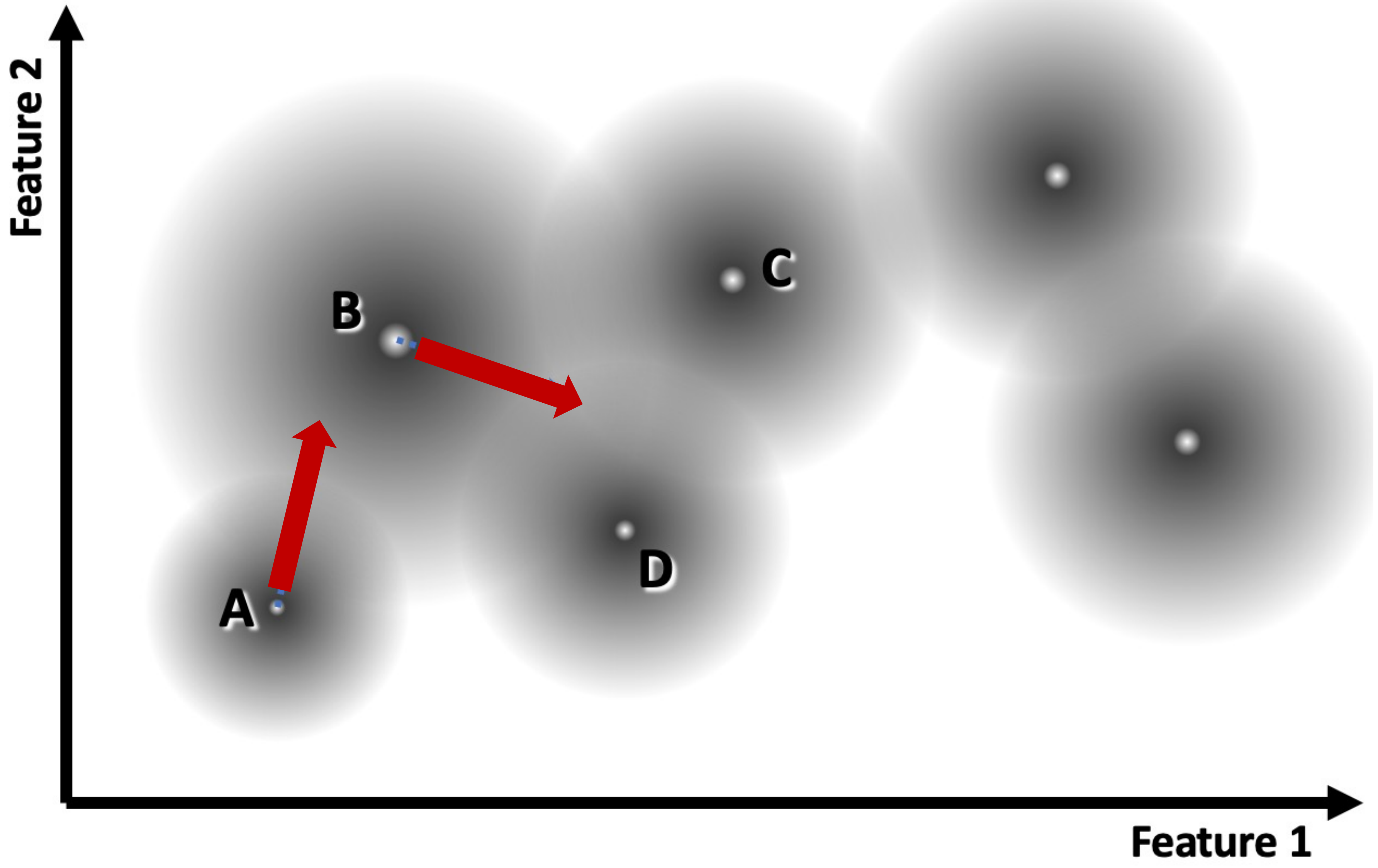
Dangerous Case

BIG CHANGE

Ask for **confirmation** or **evidence** before you make a big change.

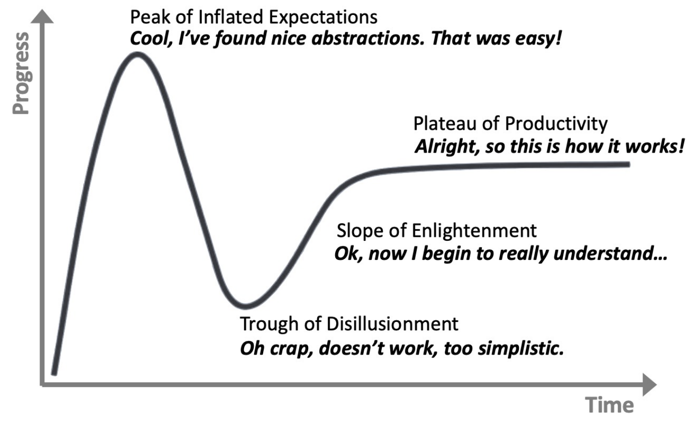
... and then potentially **change** the overall set of abstractions.

Dealing with Feedback – Gravity Model of Evolution



You gotta be willing to rethink "everything" as new data comes in.

Hopefully doesn't happen too often …



I NEED YOUR FEEDBACK

Encourage Feedback

Proof it by acting on it in a timely manner.

Dealing with Feedback

Unhelpful Case

- **Too Complicated** as a general catch-all
  - Actually bad abstractions – change.
  - More flexible – demonstrate people why it's useful
  - Learning vs. steady-state – explain and teach (see the tradeoff in discussed earler)
  - Status Quo vs. Future Needs – explain the goals and business direction
- **Superficial:** often because people didn't want to engage with the system but felt obliged to provide feedback (QA Team).
- **Unfair, unconstructive:** like superficial, but with personal pissedness. "I should have been consulted, I wasn't, so this sucks."

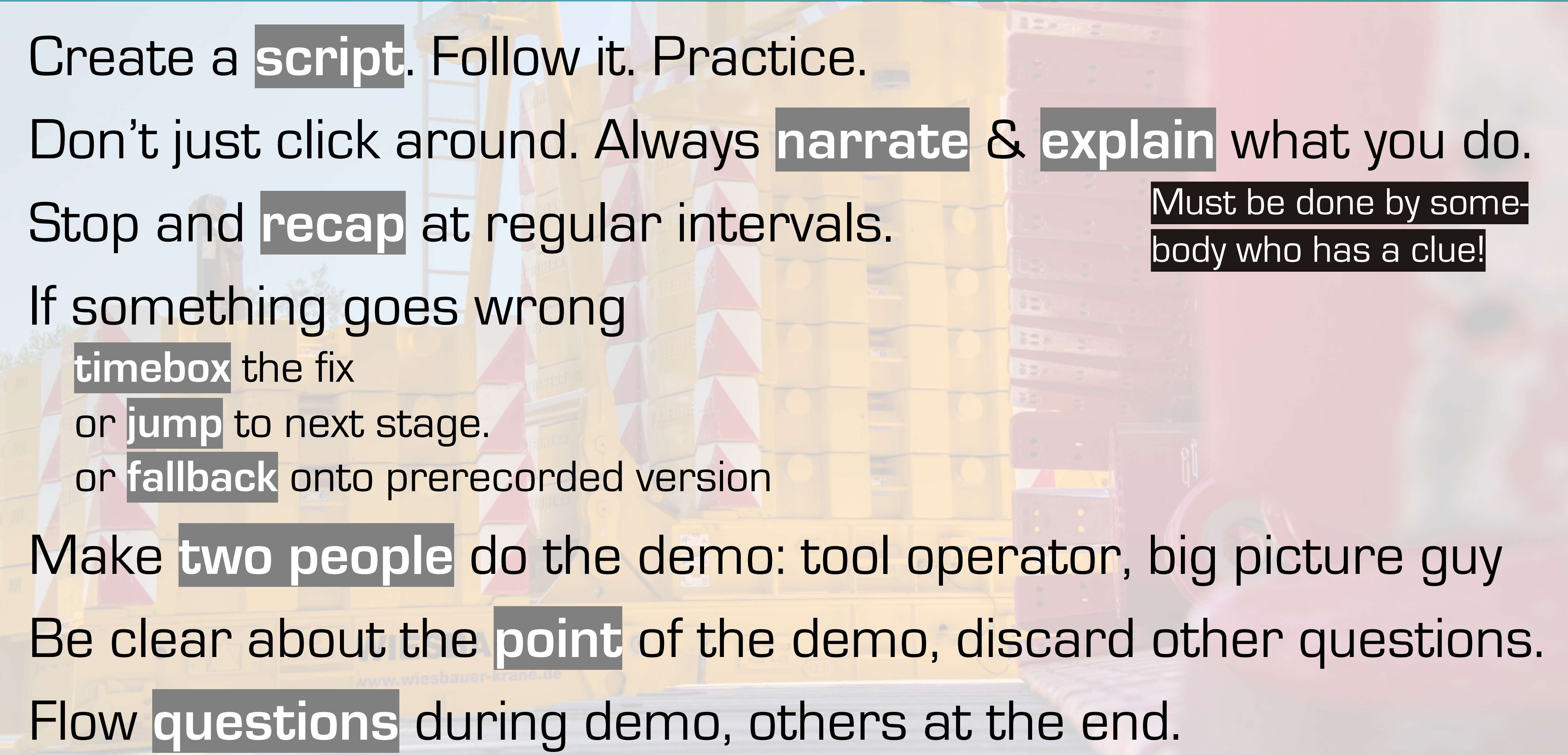
Great Demos

WIESBAUER
www.wiesbauer-krane.de
max. 200t
max. 440t
max. 160t
5,9t
13,5t
LIEBHERR

Great Demos

Don't Improvise!

Great Demos



Create a **script**. Follow it. Practice.

Don’t just click around. Always **narrate** & **explain** what you do.

Stop and **recap** at regular intervals.

Must be done by some-
body who has a clue!

If something goes wrong

- **timebox** the fix
- or **jump** to next stage.
- or **fallback** onto prerecorded version

Make **two people** do the demo: tool operator, big picture guy

Be clear about the **point** of the demo, discard other questions.

Flow **questions** during demo, others at the end.

Great Demos

Oh, and …

Don’t Improvise!

Writing – Document Structure

+ Examples

The pattern format is a great guide.

Context. Where are we coming from, where does the problem we want to solve occur?

Problem. What are we trying to solve or fix with what we describe in the text?

Forces. What influences govern the way in which we plan to solve the problem?

Solution. How in general are we approaching the problem and what is the 10,000-foot view of the solution?

Details. What are the relevant details of the solution, things that have to be kept in mind or addressed specifically?

Trade-offs. What are the pros and cons of the solution, ideally connecting to the forces and potential alternative solutions?

Resulting context. Where does this leave us; what do we do next?

Writing – Smaller Structures

One **idea** per paragraph. Single level of detail per paragraph.

First the **big picture**. Then the details. Separately.

Make **reasoning** transparent, make **assumptions** explicit.

Justify **claims**.

Say explicitly when you change **aspect** or **viewpoint**.

Don't needlessly use **synonyms** for important concepts.

Use **bullet points** judiciously. Don't emulate powerpoint.

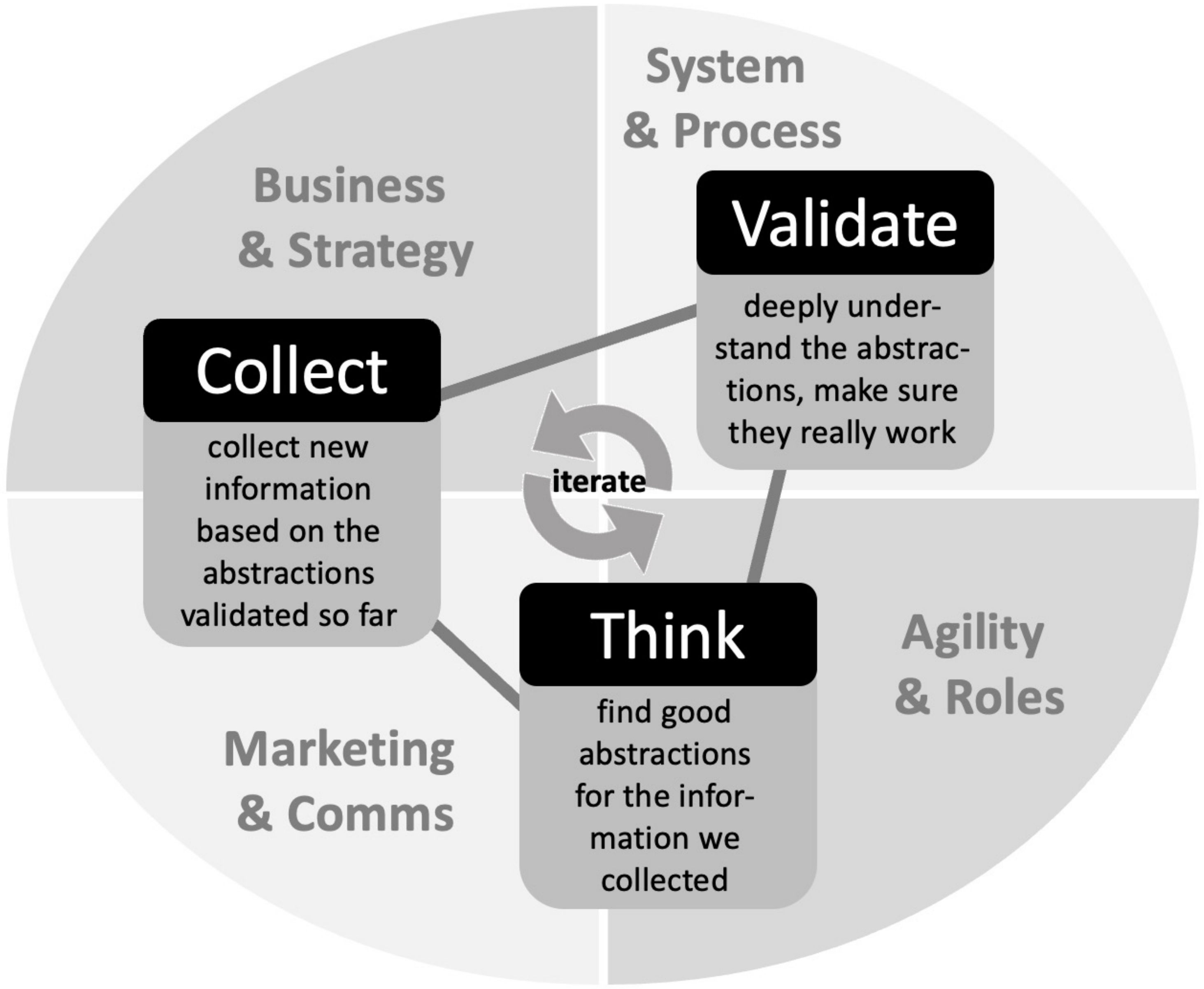
Shorter sentences are usually better sentences.

Give your document time. Reread. Edit.

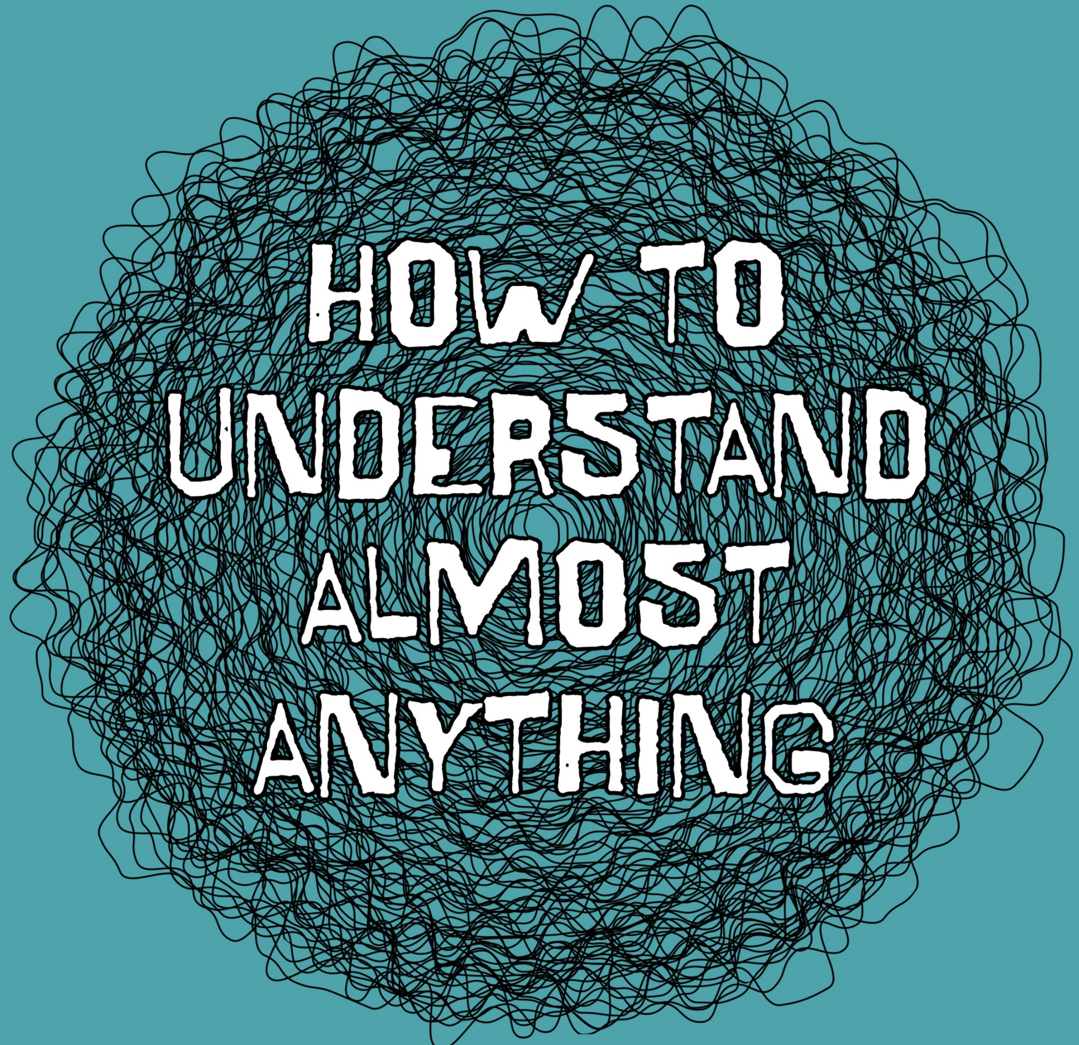
VI

OUTRO

Additional Considerations


System & Process
Business & Strategy
Validate
deeply under-stand the abstrac-tions, make sure they really work
Collect
collect new information based on the abstractions validated so far
iterate
Think
find good abstractions for the infor-mation we collected
Agility & Roles
Marketing & Comms

oop 2024
software meets business



DOMAIN ANALYSIS FOR PRACTITIONERS

Based on the book of the same name:

http://voelter.de/htuaa

There's a discount code for the PDF version at Leanpub:

https://leanpub.com/markusvoelter-htuaa/c/oop23
(expires end of Feburary)

Ping me: http://voelter.de/hello