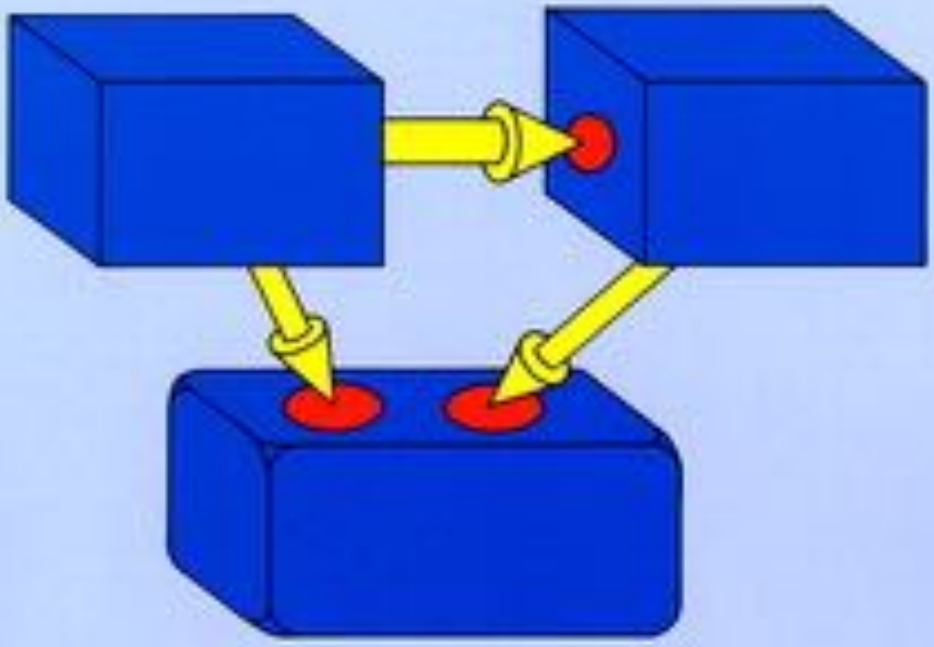
Grow Your Personal Design Heuristics

Rebecca Wirfs-Brock

Designing Object-Oriented Software



Rebecca Wirfs-Brock
Brian Wilkerson
Lauren Wiener

1990

cover art by Phil Brock

Za'atar-Roasted Broccoli Salad

with Fregola Sarda, Pecorino Cheese & Tahini Dressing

PREP TIME: 15 minutes
COOK TIME: 25-35 minutes
SERVINGS: 2

For tonight's Middle Eastern-inspired pasta salad, we're roasting broccoli with za'atar (a traditional blend of spices including oregano, thyme and sumac). The broccoli's herbaceous, savory flavor is a perfect match for toasted fregola sarda pasta. Crunchy roasted almonds, Pecorino cheese and hard-boiled egg give the salad plenty of hearty texture—all brought together with a bright, creamy tahini dressing.

BLUE APRON WINE PAIRING:
Fabre Minervois, 2015
Order wine and view other perfect pairings at blueapron.com.

Ingredients

- 2 CAGE-FREE FARM EGGS
- 1 clove GARLIC
- ½ cup FREGOLA SARDA PASTA
- 1 LEMON
- 1 lb BROCCOLI
- 1 RED ONION
- 1 bunch MINT

KNICK KNACKS:

- 2 Tbsp ROASTED ALMONDS
- 2 oz PECORINO CHEESE
- 2 Tbsp TAHINI
- 2 tsp ZA'ATAR

Blue Apron

Download our iOS app or log in to blueapron.com for how-to videos and supplier stories.

Nothing ever goes exactly by the book

1 Prepare the ingredients:

- ☐ Preheat the oven to 450°F.
- ☐ Wash and dry the fresh produce.
- ☐ Heat a small pot of water to boiling on high.
- ☐ Heat a large pot of salted water to boiling on high.
- ☐ Cut the broccoli into bite-sized florets.
- ☐ Peel and thinly slice the onion.
- ☐ Peel and mince the garlic; using the flat side of your knife, smash until it resembles a paste (or use a zester).
- ☐ Quarter and deseed the lemon.
- ☐ Roughly chop the almonds.
- ☐ Using a fork, crumble the cheese into small pieces.
- ☐ Pick the mint leaves off the stems; discard the stems.

2 Cook & peel the eggs:

- ☐ Carefully add the **eggs** to the small pot of boiling water and cook for exactly 9 minutes.
- ☐ Drain and rinse under cold water for 30 seconds to 1 minute to stop the cooking process.
- ☐ When cool enough to handle, carefully peel the cooked eggs.
- ☐ Transfer to a cutting board and thinly slice into rounds. Season with salt and pepper.

3 Roast the broccoli & onion:

- ☐ While the eggs cook, place the **broccoli** and **onion** on a sheet pan.
- ☐ Drizzle with olive oil and season with salt, pepper and the **za'atar**; toss to thoroughly coat.
- ☐ Arrange in a single, even layer and roast 20 to 22 minutes, or until browned and tender when pierced with a fork. Remove from the oven.

4 Make the dressing:

- ☐ While the broccoli and onion roast, in a bowl, combine the **tahini**, **the juice of 2 lemon wedges**, **2 tablespoons of water** and **as much of the garlic paste as you'd like**.
- ☐ Slowly whisk in **1 ½ tablespoons of olive oil** until well combined. Season with salt and pepper to taste.

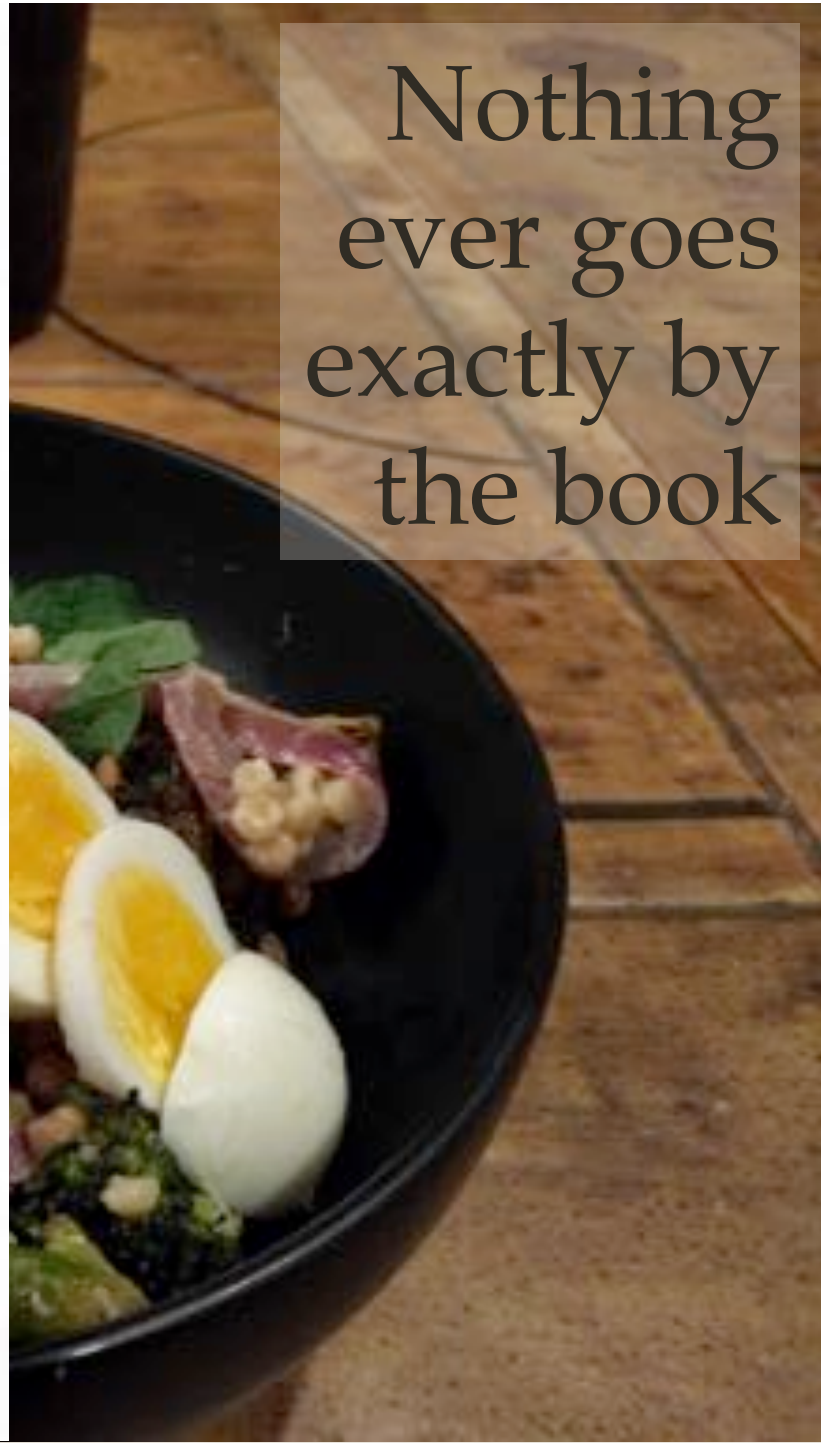
5 Cook the pasta:

- ☐ While the broccoli and onion continue to roast, add the **pasta** to the large pot of boiling water and cook 14 to 17 minutes, or until tender.
- ☐ Turn off the heat. Drain thoroughly and return to the pot.

6 Finish & plate your dish:

- ☐ To the pot of **cooked pasta**, add the **roasted broccoli and onion**, **almonds**, **cheese**, **dressing**, **the juice of the remaining lemon wedges** and a drizzle of olive oil.
- ☐ Stir to thoroughly combine; season with salt and pepper to taste.
- ☐ Divide the finished salad between 2 dishes.
- ☐ Top with the **sliced eggs**.
- ☐ Garnish with the **mint** (tearing just before adding). Enjoy!

Share your photos **#blueapron**



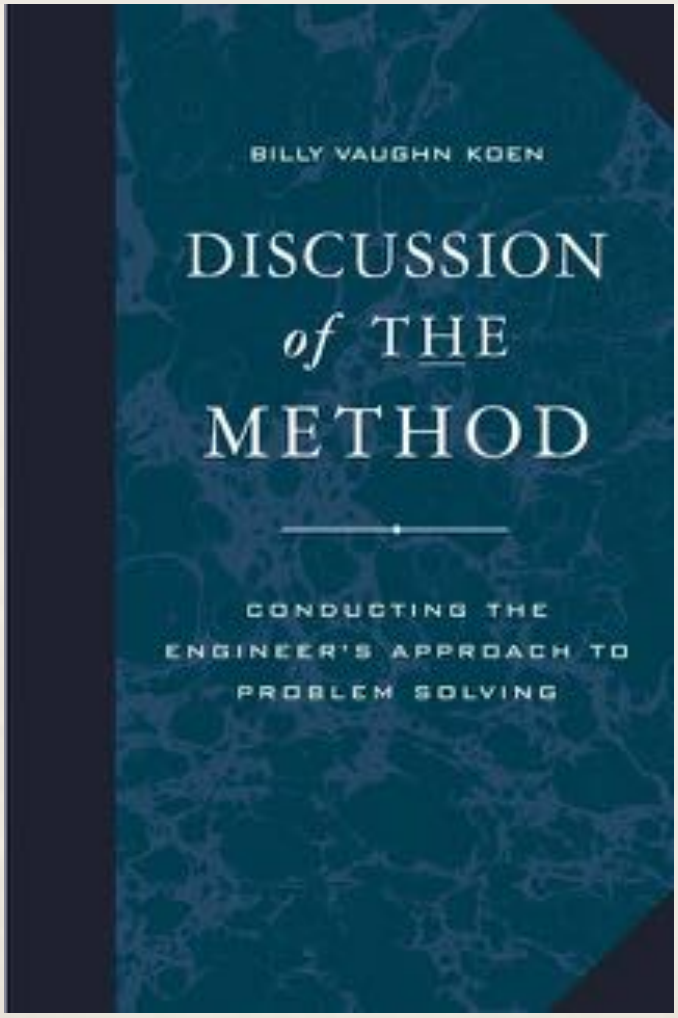
... there is no substitute for learning from your own experience & personal reflection

"any approach to problem solving, learning, or discovery that employs a practical method not guaranteed to be optimal or perfect, but sufficient for the immediate goals." —Wikipedia

Heuristic / euristic

"anything that provides a plausible aid or direction in the solution of a problem but is in the final analysis unjustified, incapable of justification, and potentially fallible."

—Billy Vaughn Koen



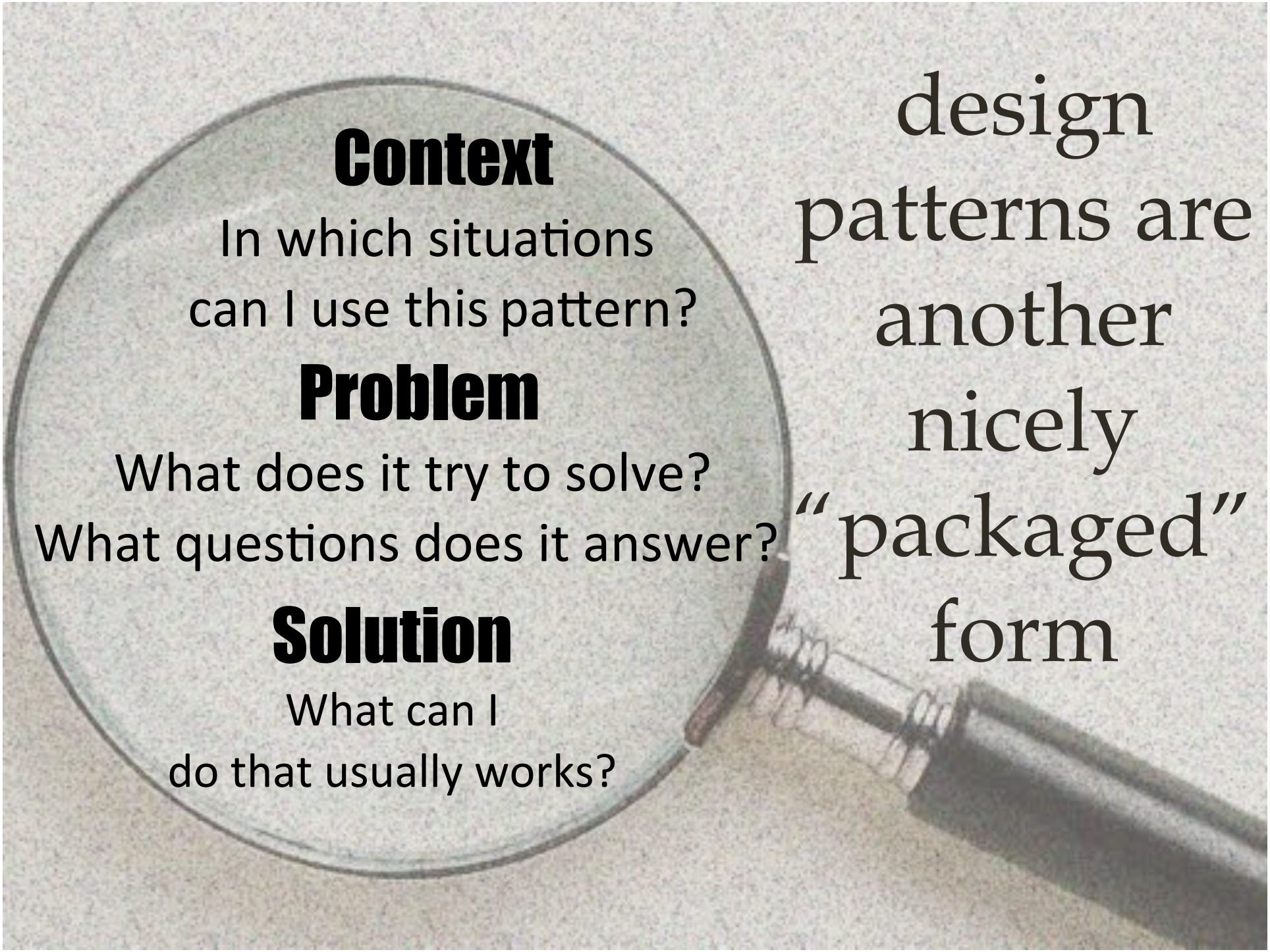
A Few General Engineering Heuristics by Billy

Solve problems by successive approximations.

Always give an answer.

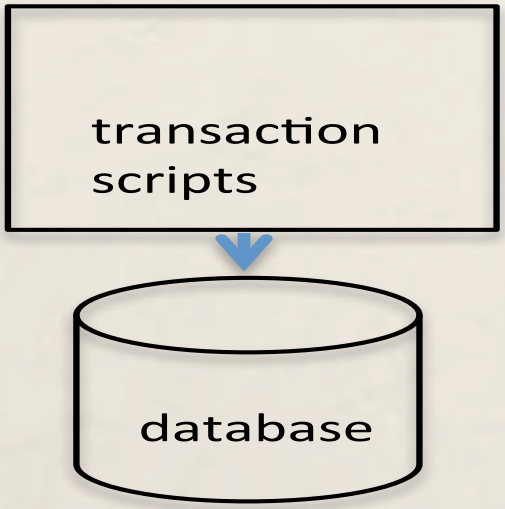
Use feedback to stabilize your design.

Always give yourself a chance to retreat.

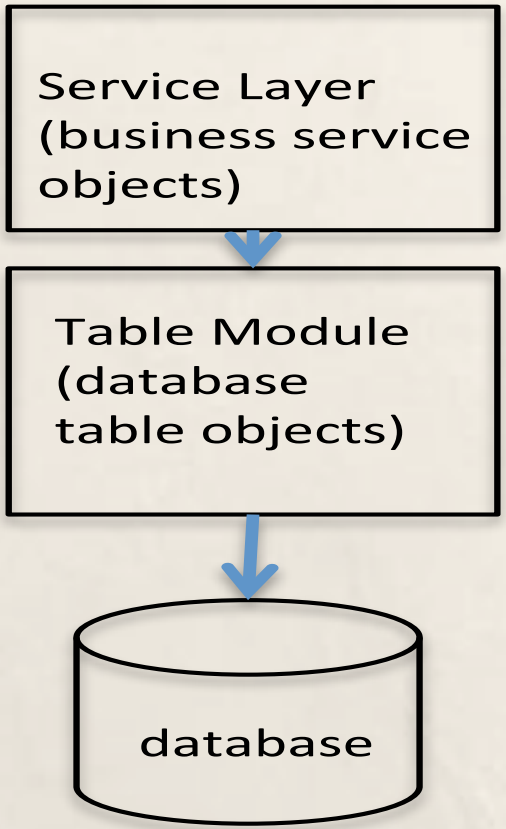

Context
In which situations
can I use this pattern?
Problem
What does it try to solve?
What questions does it answer?
Solution
What can I
do that usually works?
design
patterns are
another
nicely
“packaged”
form

1.
Heuristics to Solve a Design Problem

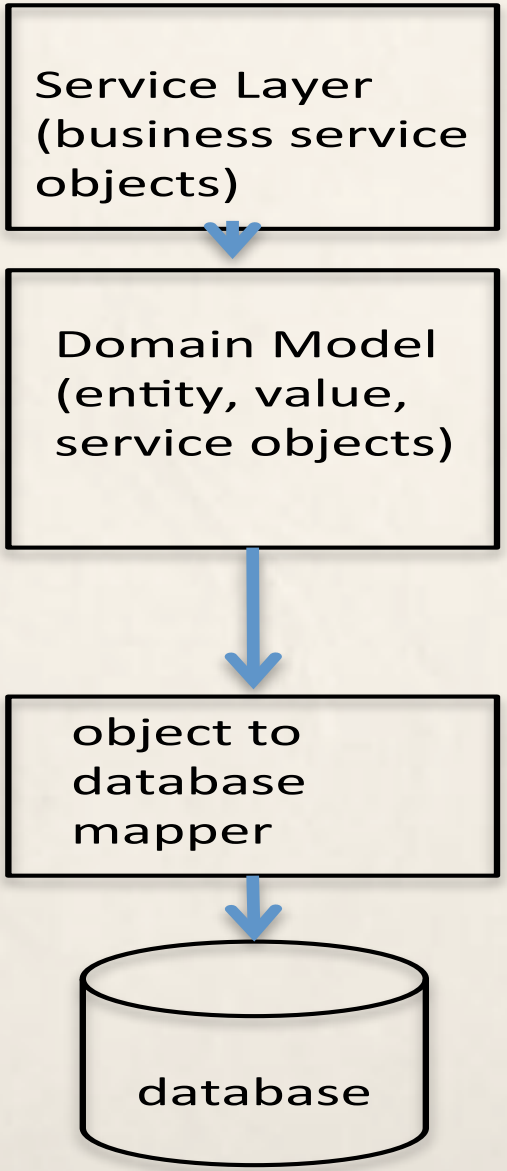
a. Transaction Script Pattern



b. Table Module Pattern



c. Domain Model Pattern



Three Approaches for Structuring the Domain Layer

Patterns of Enterprise Application Architecture

a. Transaction Script Pattern

transaction scripts

Heuristic:
Use for simple apps and data

b. Table Module Pattern

Service Layer (business service objects)

Heuristic:
Use for complex existing data and logic applying to multiple "rows"

database

c. Domain Model Pattern

Service Layer (business service objects)

Heuristic:
Use for complex logic and accept cost of db mapping

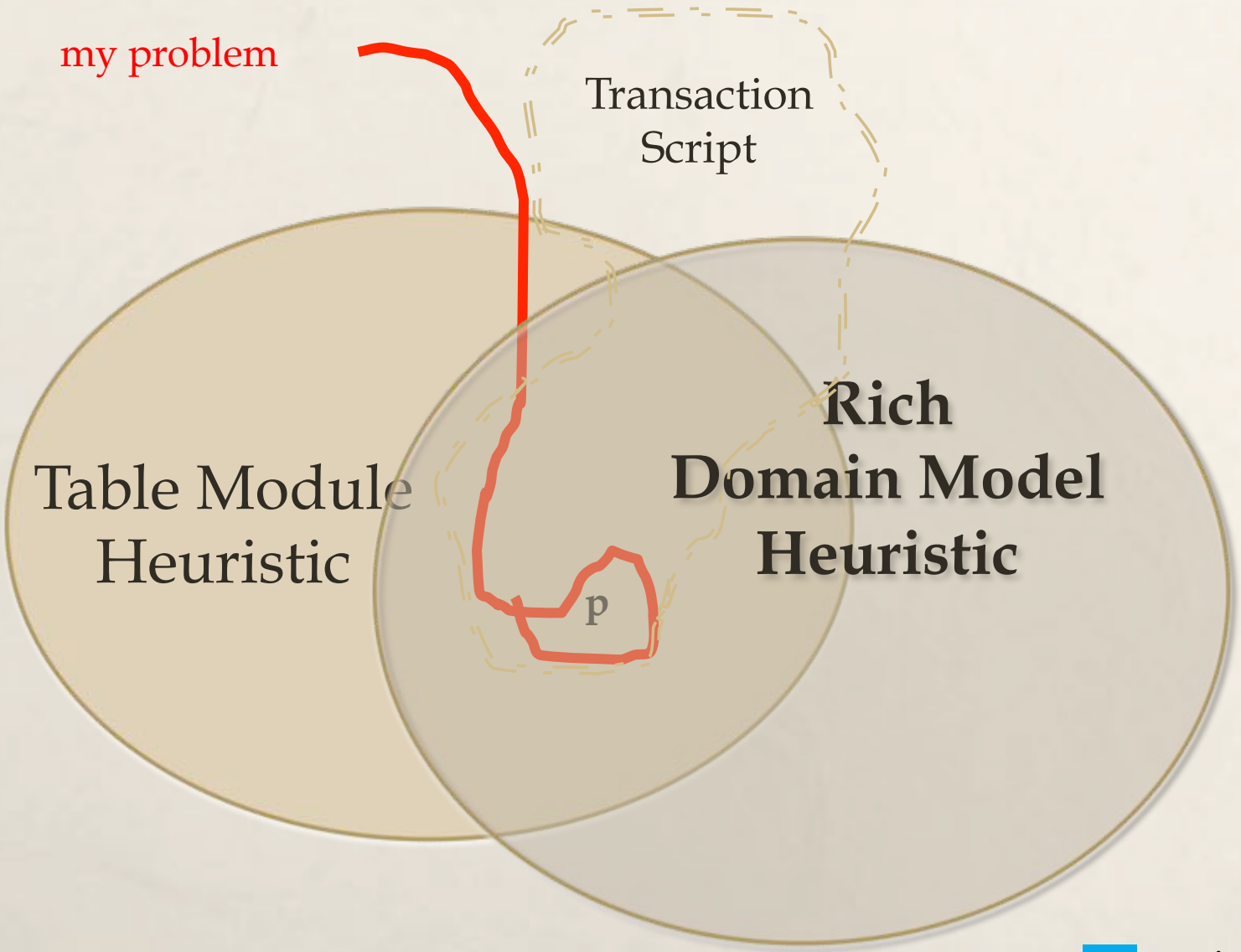
object to database mapper

database

Three Approaches for Structuring the Domain Layer

Patterns of Enterprise Application Architecture

Solvable problems



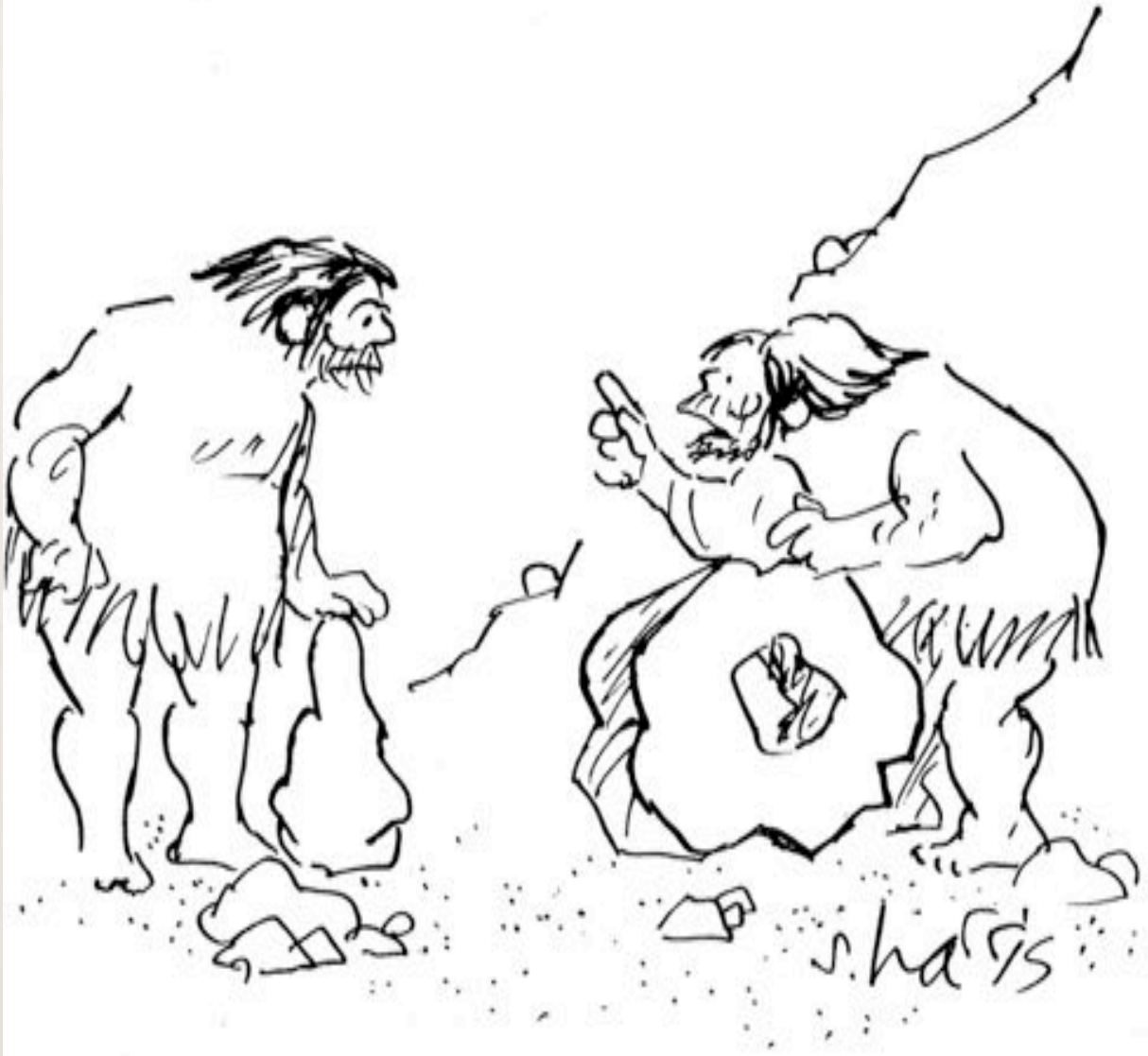
@rebeccawb

But Martin, wh about CQRS tures or micro-service architectures?

NO FAIR

@rebeccawb

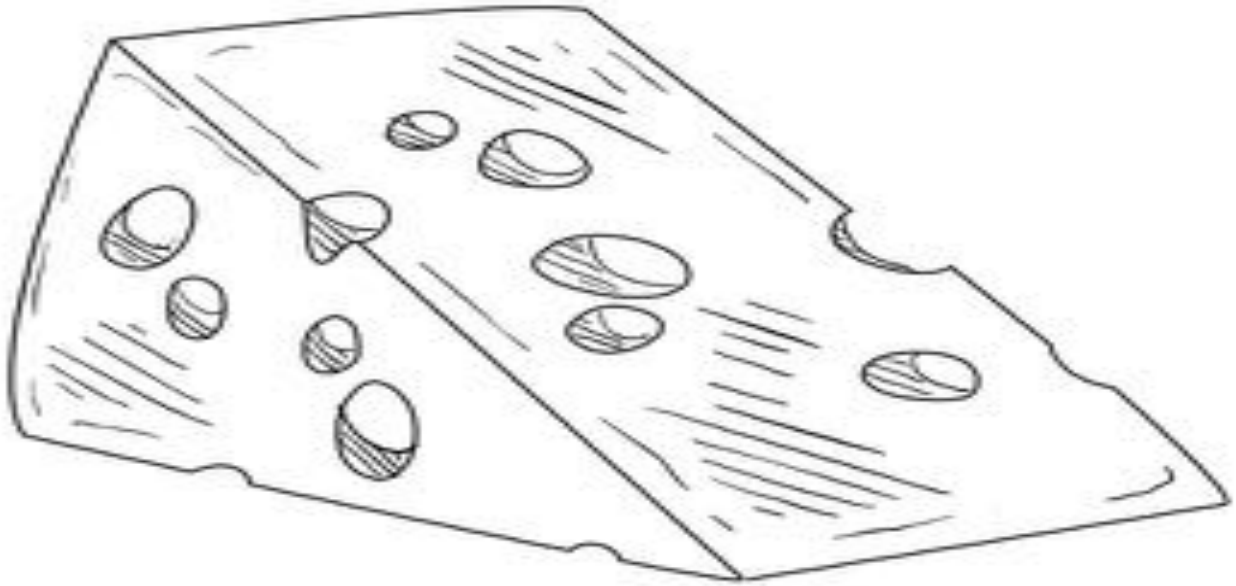
Our state-of-the-art is constantly progressing



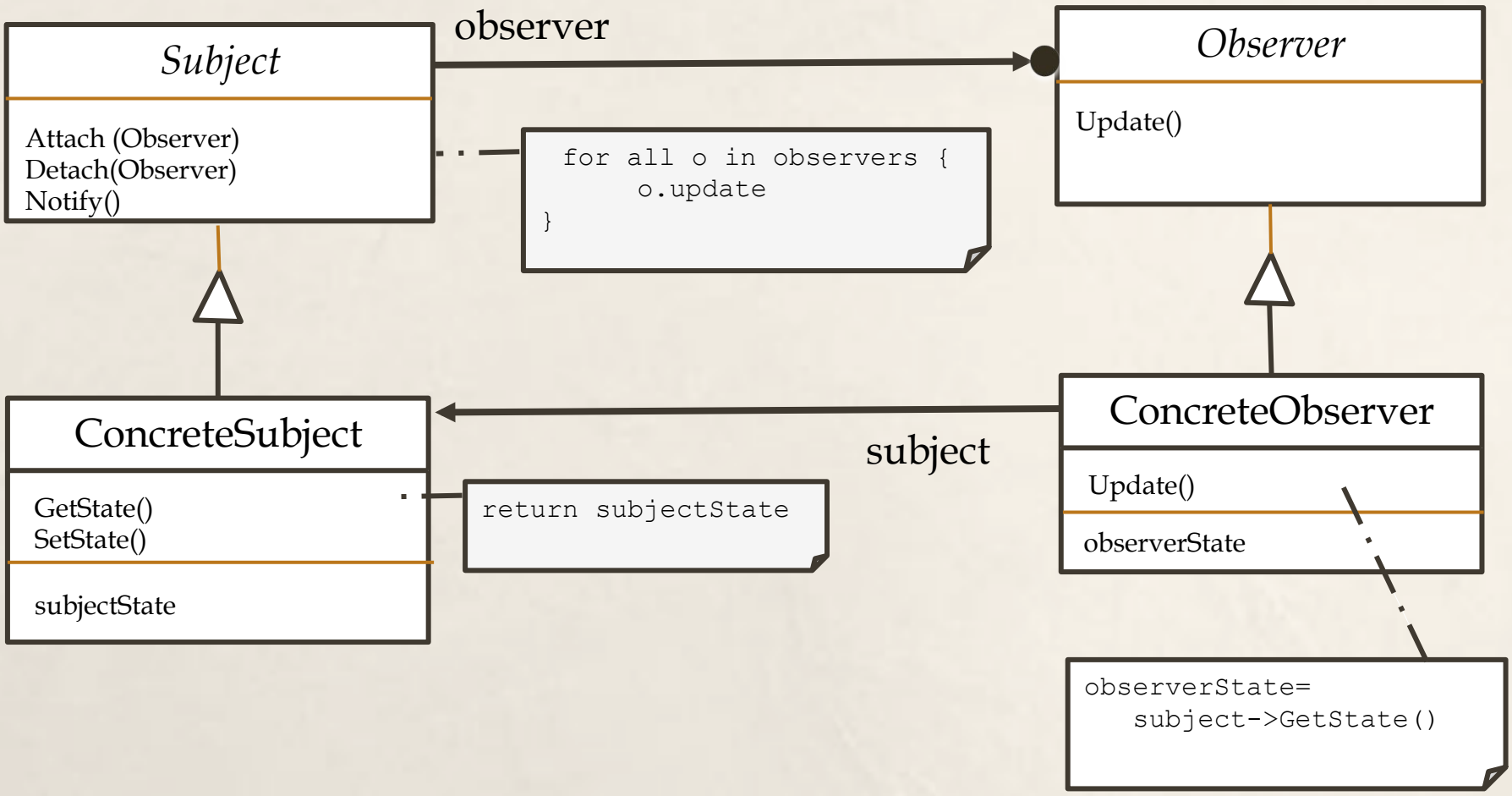
…don't judge an older system (or its designers) based on today's heuristics.


@rebeccawb

Patterns are never complete…

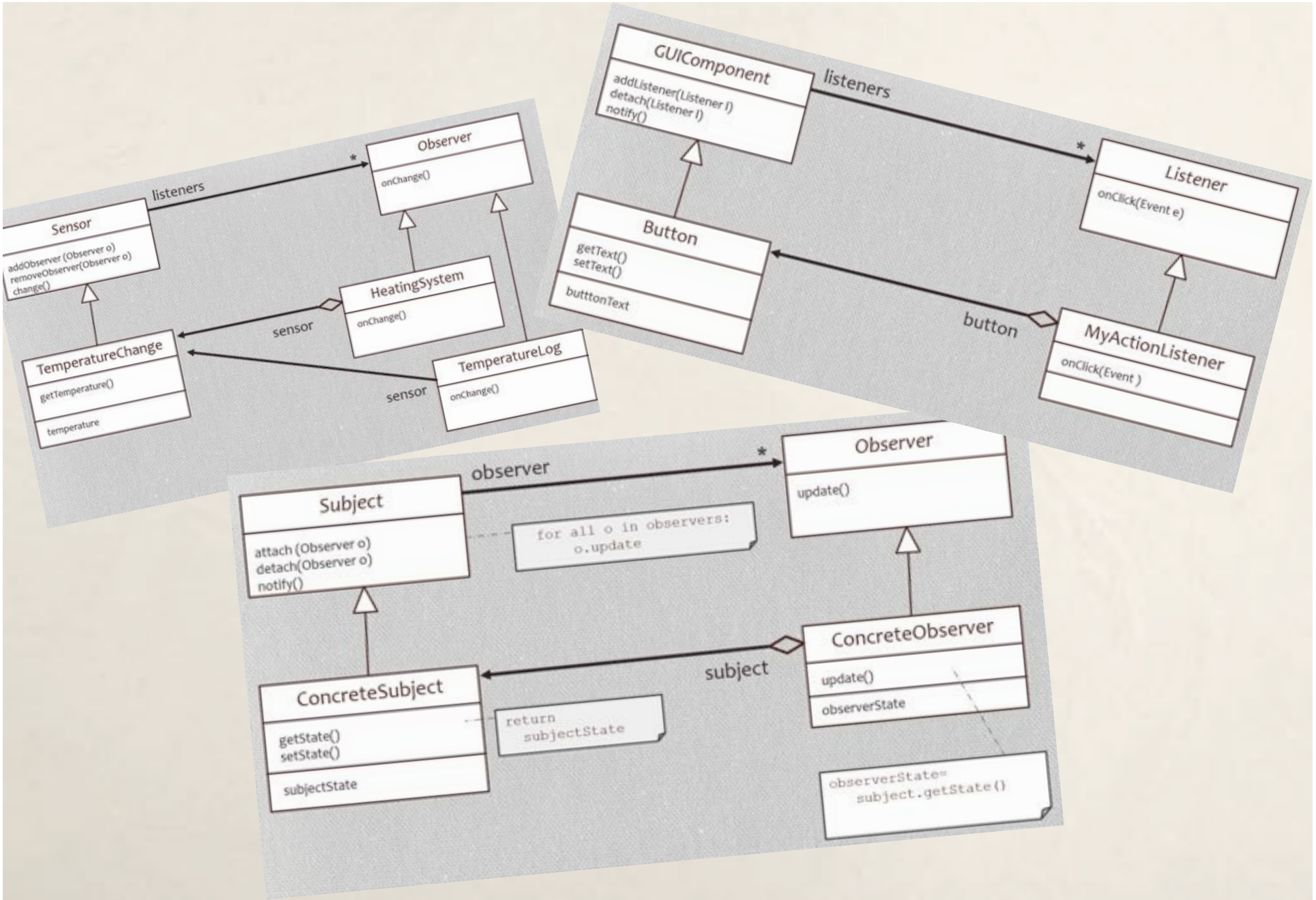


…pattern authors write about what they know and what they see that works



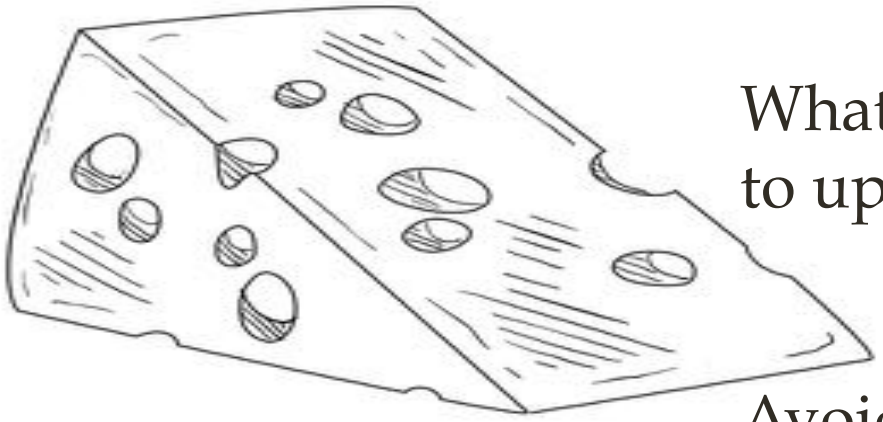
The Observer Pattern

Design Patterns Gamma et al., 1995



How should we really design this?

Our own heuristics have to fill in the gaps…



What info to send with a notify?

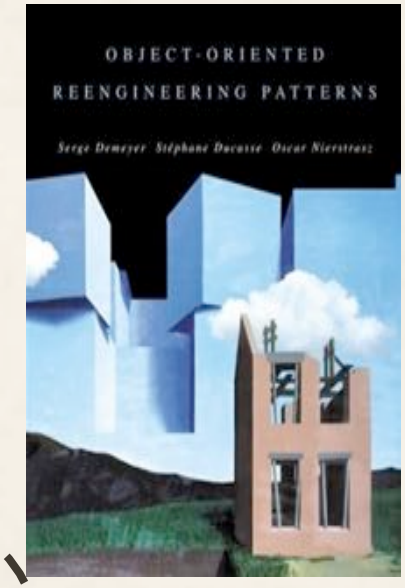
How should I handle an indeterminate number of observers?

What are appropriate responses to update notices?

Avoiding infinite react/change/react loops….

2.
Heuristics to Guide Use of Other Heuristics

First Contact Patterns



System experts

Talk with end users

Pattern 3.1: Chat with the Maintainers

Talk with developers

Pattern 3.4: Interview During Demo

Talk about it

Software System

Verify what you hear

Read it

Pattern 3.2: Read All the Code in One Hour

Read about it

Pattern 3.3: Skim the Documentation

Compile it

Pattern 3.5: Do a Mock Installation

Each chapter in *Object-Oriented Reengineering Patterns* is a small language

@rebeccawb

3.
Heuristics that Determine our Attitude and Behavior

Manifesto for Software Craftsmanship

Raising the bar.

As aspiring Software Craftsmen we are raising the bar of professional software development by practicing it and helping others learn the craft. Through this work we have come to value:

Not only working software,
but also **well-crafted software**

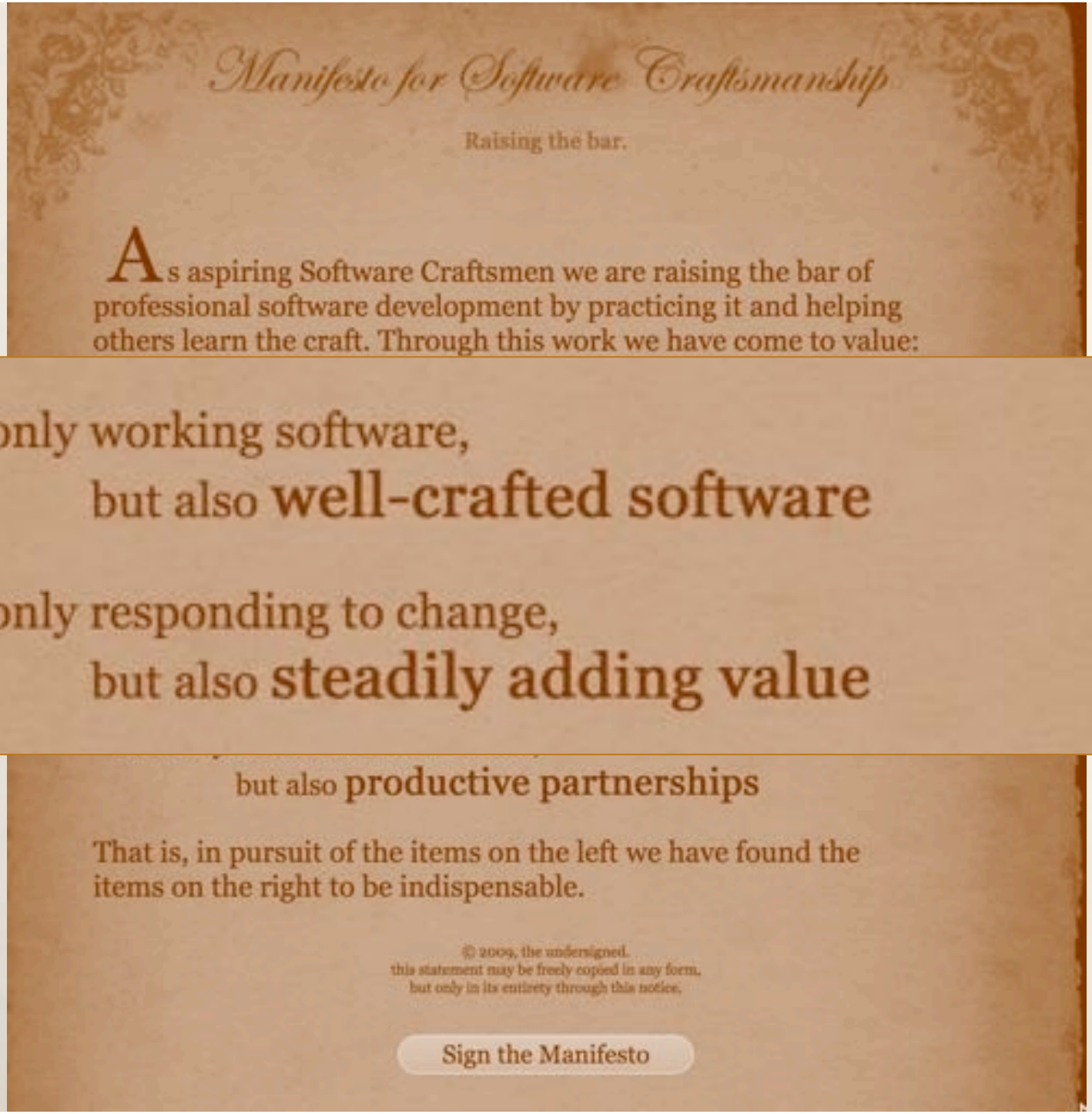
Not only responding to change,
but also **steadily adding value**

Not only individuals and interactions,
but also **a community of professionals**

Not only customer collaboration,
but also **productive partnerships**

That is, in pursuit of the items on the left we have found the items on the right to be indispensable.

Sign the Manifesto


Manifesto for Software Craftsmanship
Raising the bar.
As aspiring Software Craftsmen we are raising the bar of professional software development by practicing it and helping others learn the craft. Through this work we have come to value:
Not only working software,
but also well-crafted software
Not only responding to change,
but also steadily adding value
but also productive partnerships
That is, in pursuit of the items on the left we have found the items on the right to be indispensable.
© 2009, the undersigned.
this statement may be freely copied in any form,
but only in its entirety through this notice.
Sign the Manifesto

“Value consistency over cleverness.”

Our Values Drive Behavior

principle: acceptance of something as truth

value: a person's personal belief for or against something (in a particular context)

My Agile Design Values

Value evidence over speculation

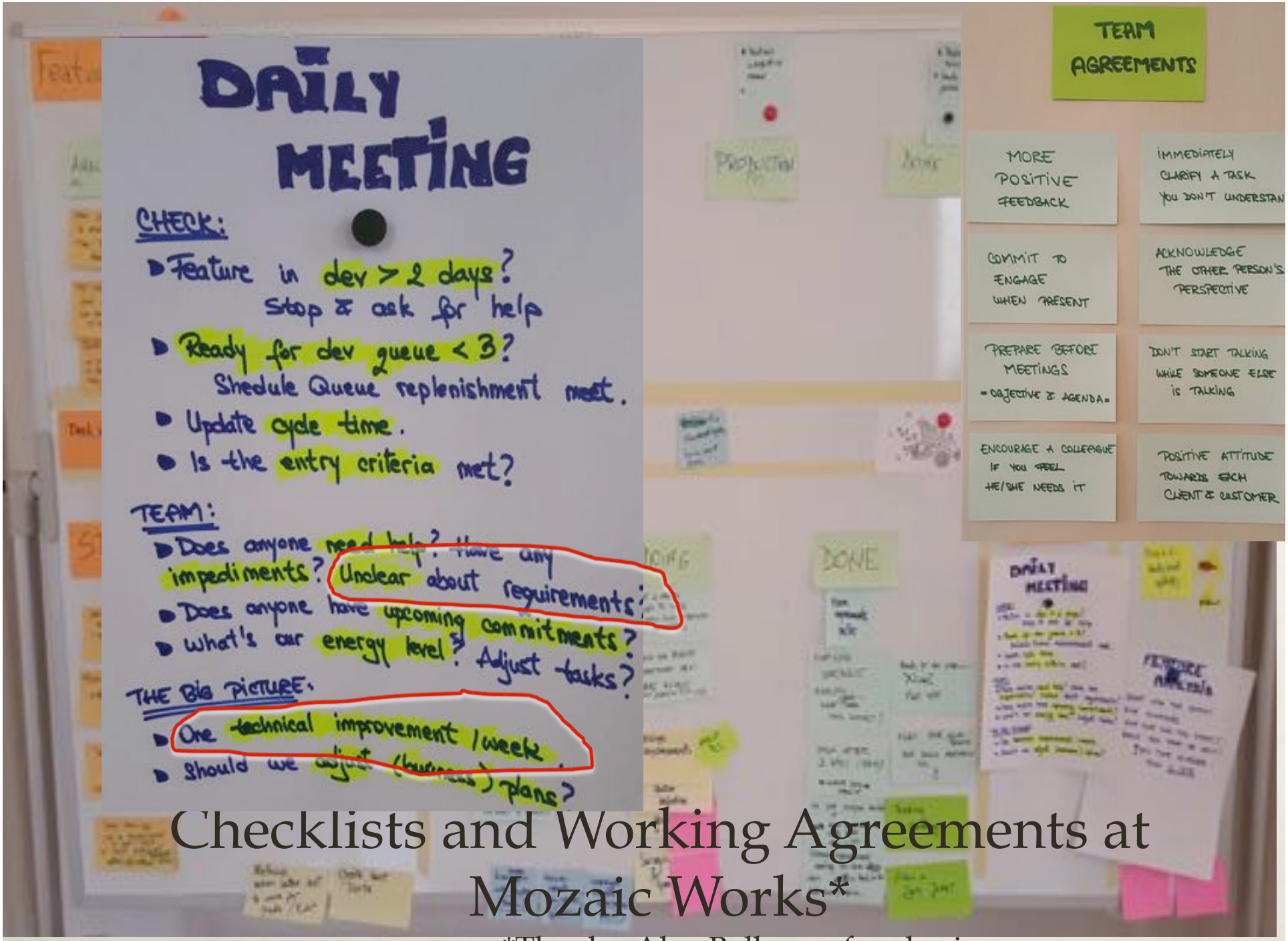
Design for what I know now

Learn, then adapt

Simple design (uncomplicated) if possible, not simplistic

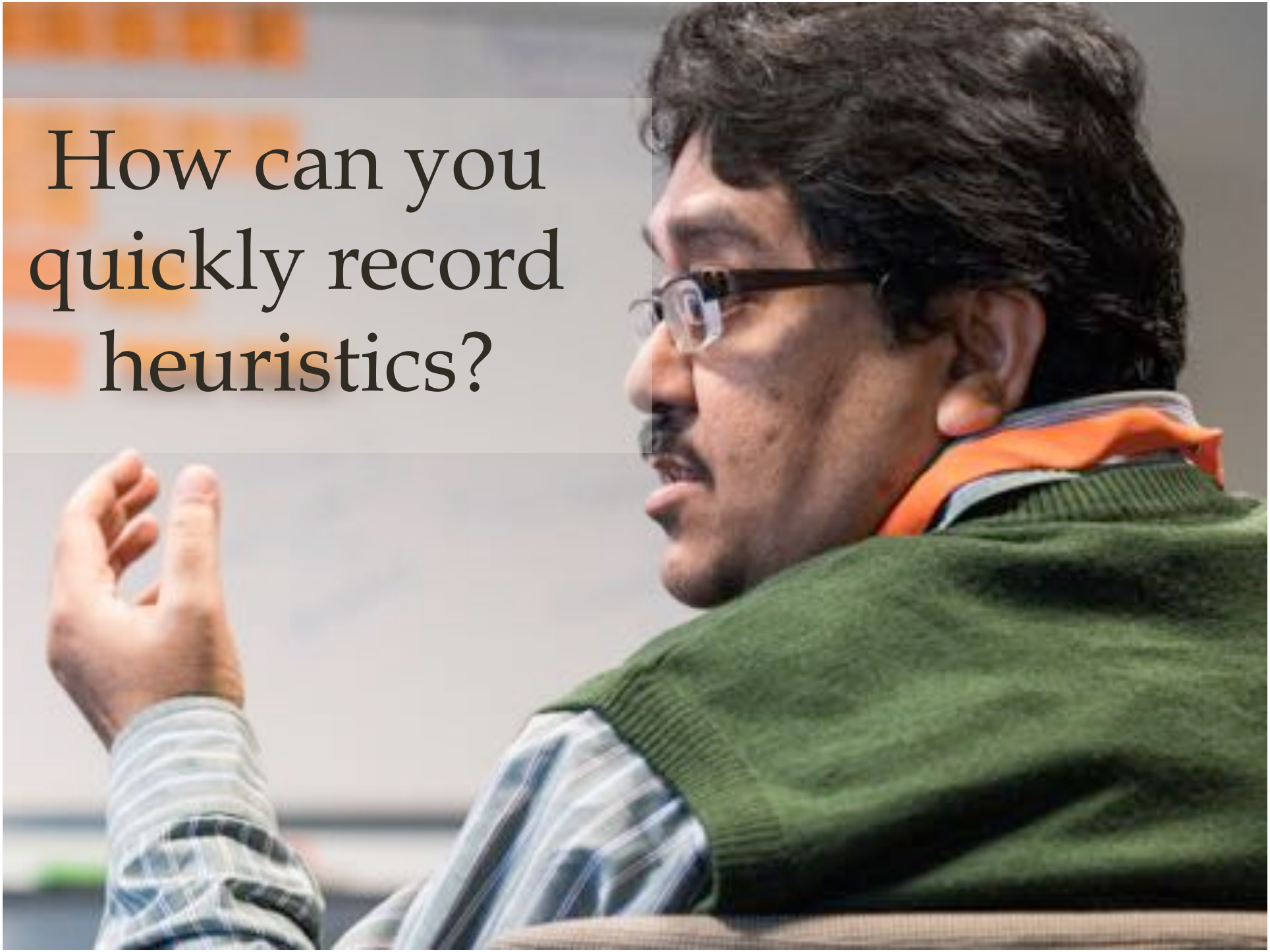
Expect the architecture to evolve

Photo courtesy of
https://www.flickr.com/photos/notbrucelee/7113385543



Checklists and Working Agreements at Mozaic Works*

*Thanks, Alex Balboaca for sharing


How can you quickly record heuristics?


Avoid
Modeling
Gotchas
Identify
Trusted Events
Happy Path +
Expected
Exception
Lazy
microservices

Say more on

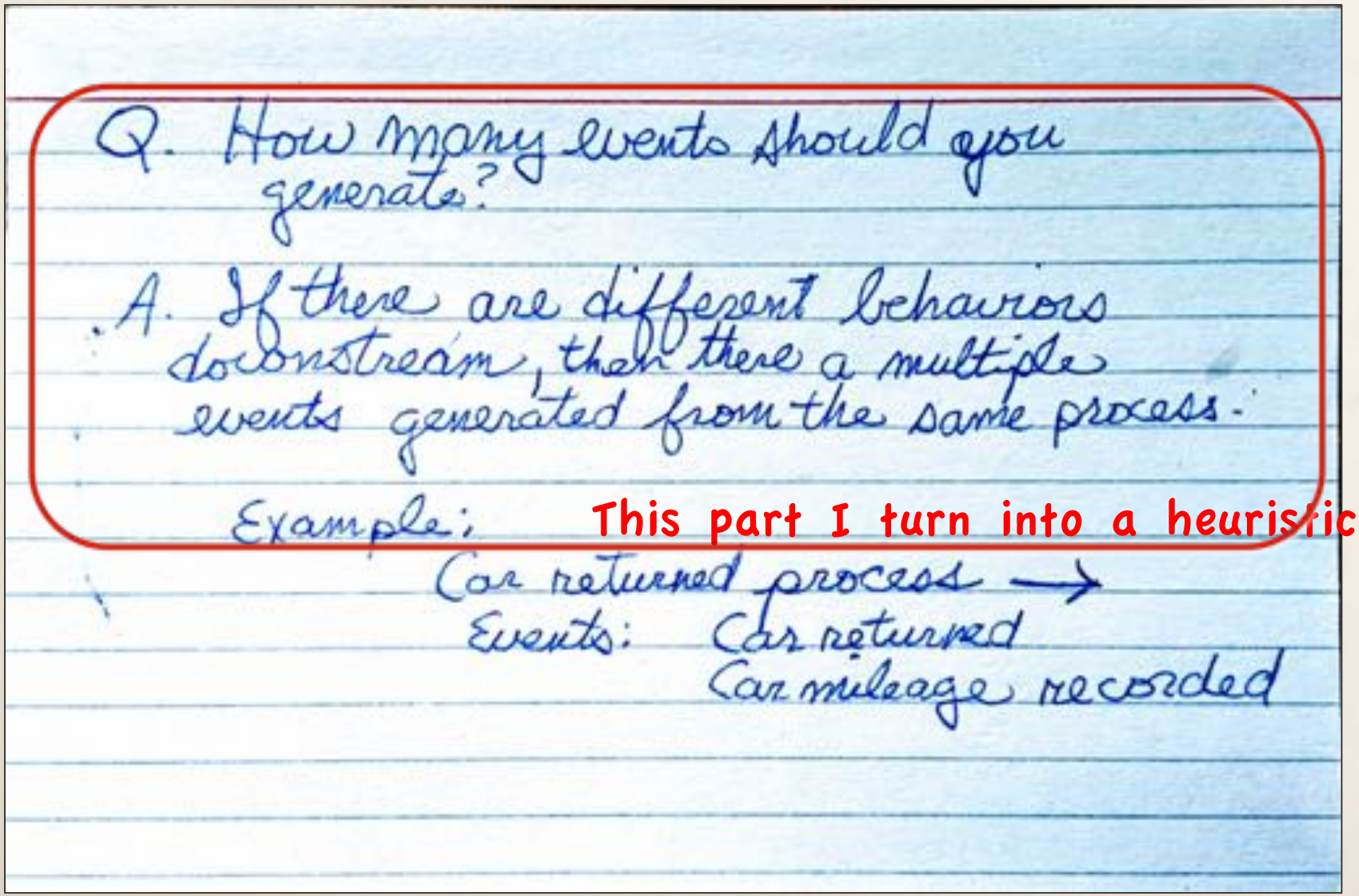
Question, Heuristic, Example (QHE) Cards

Q. When should I generate a different event?

A. If differents actors are involved, create a different event, even if the system is in the same state.

Example: Car accident reported by renter
Accident reported by agent
Accident reported by car telemetry

Question-Heuristic-Example Cards



Heuristic: Generate multiple events for a business process when downstream business processes react differently. Make each event discrete.

Heuristic Gists*

Multiple Events for a Single Process
You need to balance passing along information needed by downstream processes in a single business event with creating multiple event records, each designed to convey specific information needed by a specific downstream process.

Summary of Problem
How do you know how many events to generate from a single business process?

Summary of Solution
If processes downstream react differently, generate different discrete events. For example, handling a "rental car return" request might generate two events and event records: "car returned" and "mileage recorded." Even though the mileage is recorded at the time a car is returned, mileage could be recorded at any other time as well. It is a cleaner design to generate two events, rather than cram information into a single, overloaded "car returned" event.

*gist – the main point or part; essence. Similar to pattern thumbnails.

Not all Heuristics are Candidates for Patternhood

- ✓ Works in different contexts
- ✓ Not obvious
- * Not good candidates:
  - * Specific to a particular situation
  - * Can be stated simply:

    do **X** when you see **Y**....

Techniques for Actively Growing and Sharing Your Heuristics

Challenge Others' Heuristics

How Big Should a Microservice Be?

"…small enough and no smaller"

—Sam Newman

*"In my view a single deployable service should be **no bigger than a bounded context**, but no smaller than an **aggregate.**"*

–Ben Morris

"Single Responsibility Principle: there should only be a single service impacted by a change to the definition of this data.
As a result, you'll tend to see services that aren't all that small, and probably not so many of them. In my experience, I've seen between 7 and 15 services the majority of the time."

—Udi Dahan

Sam Newman @samnewman · May 5

Now *this* is an awesome answer to "how big should Microservices be?"

> **TLDK** @tldkhatai · May 5
>
> Replying to @samnewman
>
> so big that they overlap and transform into one monolith

2 13 71

Heuristics Often Conflict…

Choose the heuristic to use from what ***you*** take to be the best option at the time you are required to choose.

How Big Should Microservices Be?

1,801 views • May 3, 2020

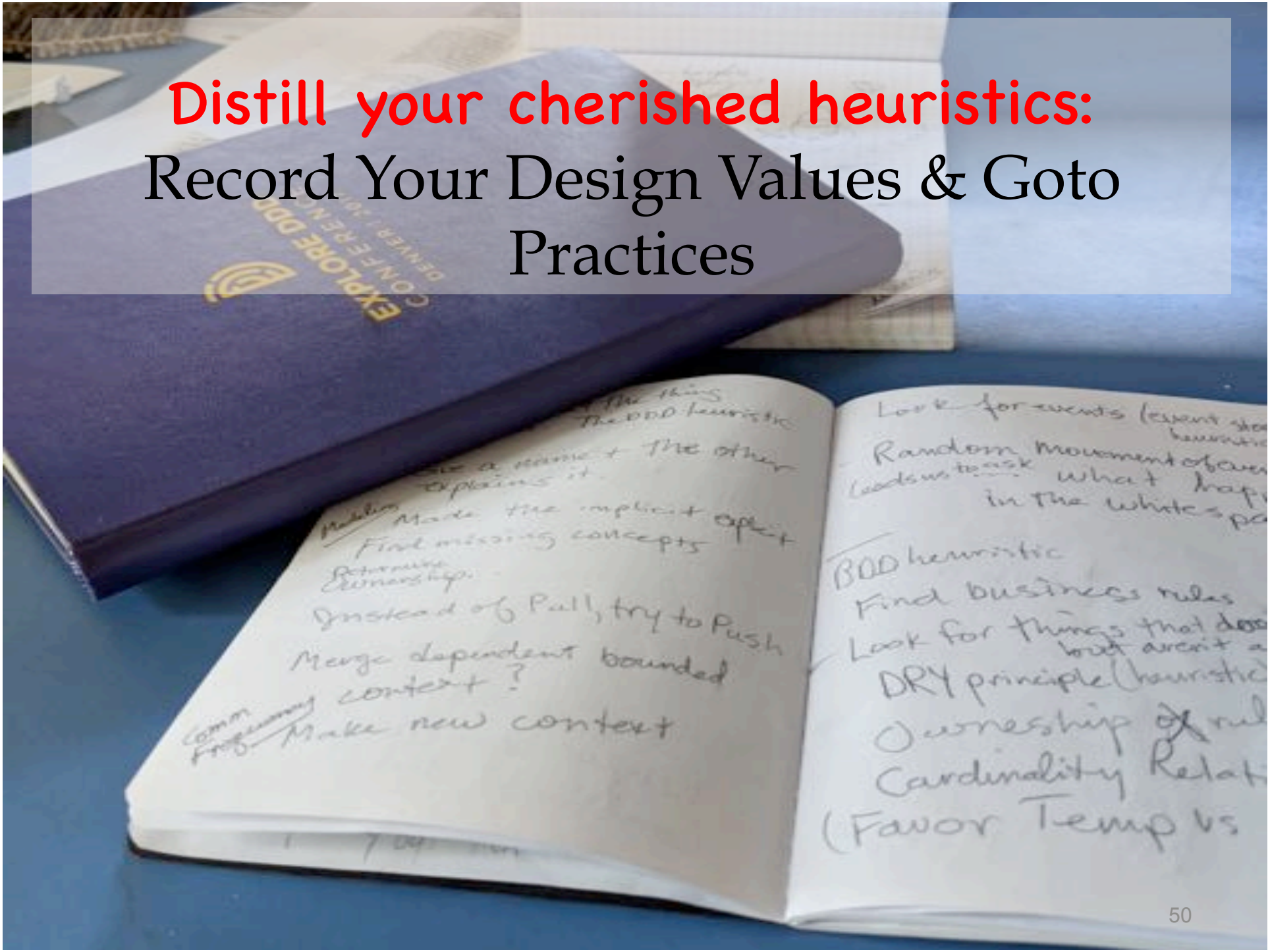
101 0 SHARE SAVE ...

Sam Newman
623 subscribers

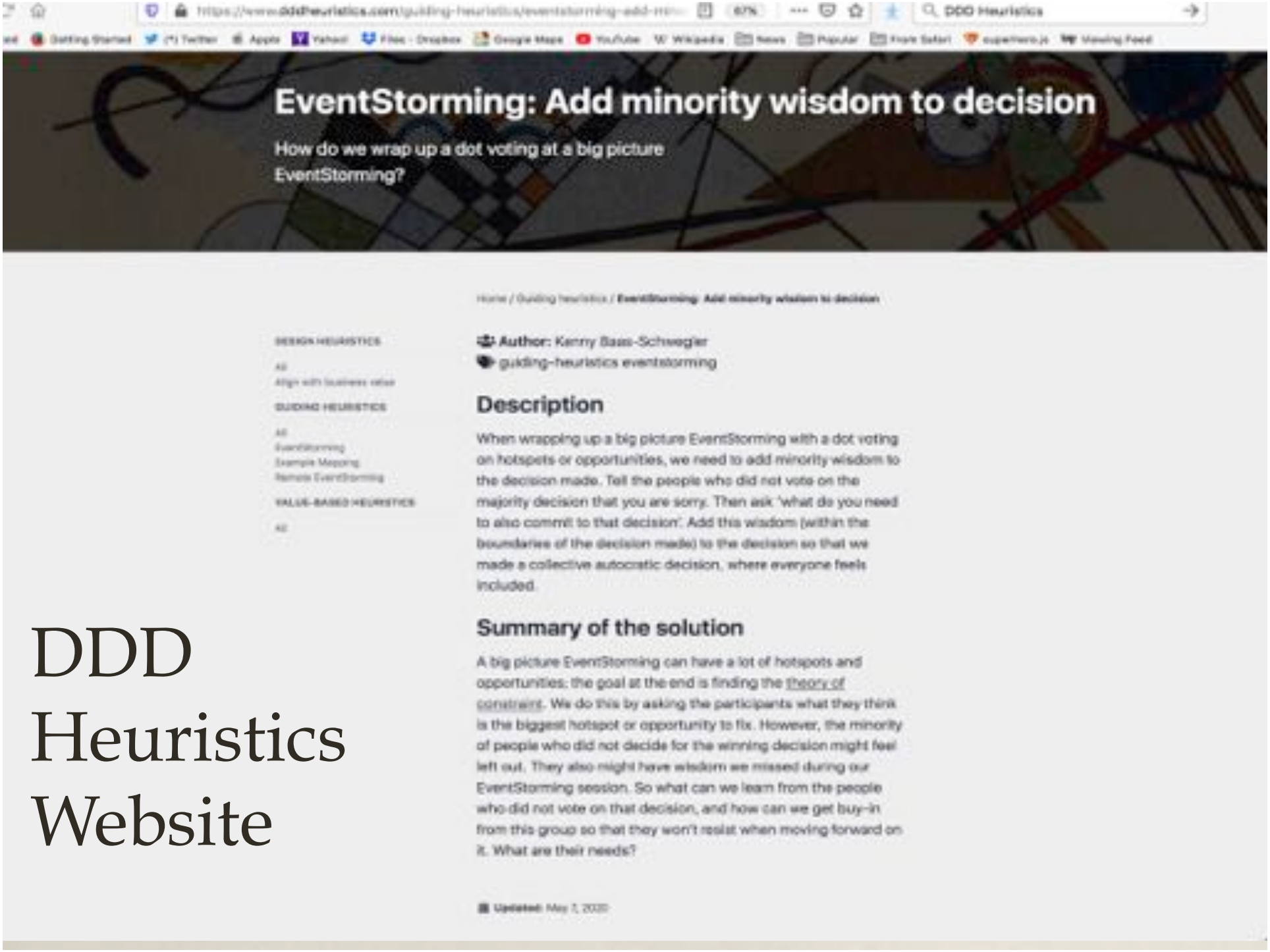
SUBSCRIBE

Another frequently asked question about microservices this week, and it's a doozy one. How big should microservices be?


Some Of Your
Heuristics Have
Aged Well!
© Can Stock Photo / Noofoo


Distill your cherished heuristics:
Record Your Design Values & Goto Practices
Look for events (event
Random movement of
What
BDD heuristic
Find business rules
Look for things that
DRY principle (heuristic
Ownership
Cardinality
(Favor Temp vs
Instead of Pull, try to Push
Merge dependent bounded context?
Make new context

DDD Heuristics Website



1. Share and compare your preferred heuristics


TIME
ROLE PLAYING
COLLABS?
Let's model!
FEW
REAL STORIES
EMOTION
GET OUT
GET THEM IN
GET HIGHER

Have a Lean Coffee Heuristics Hunting Session

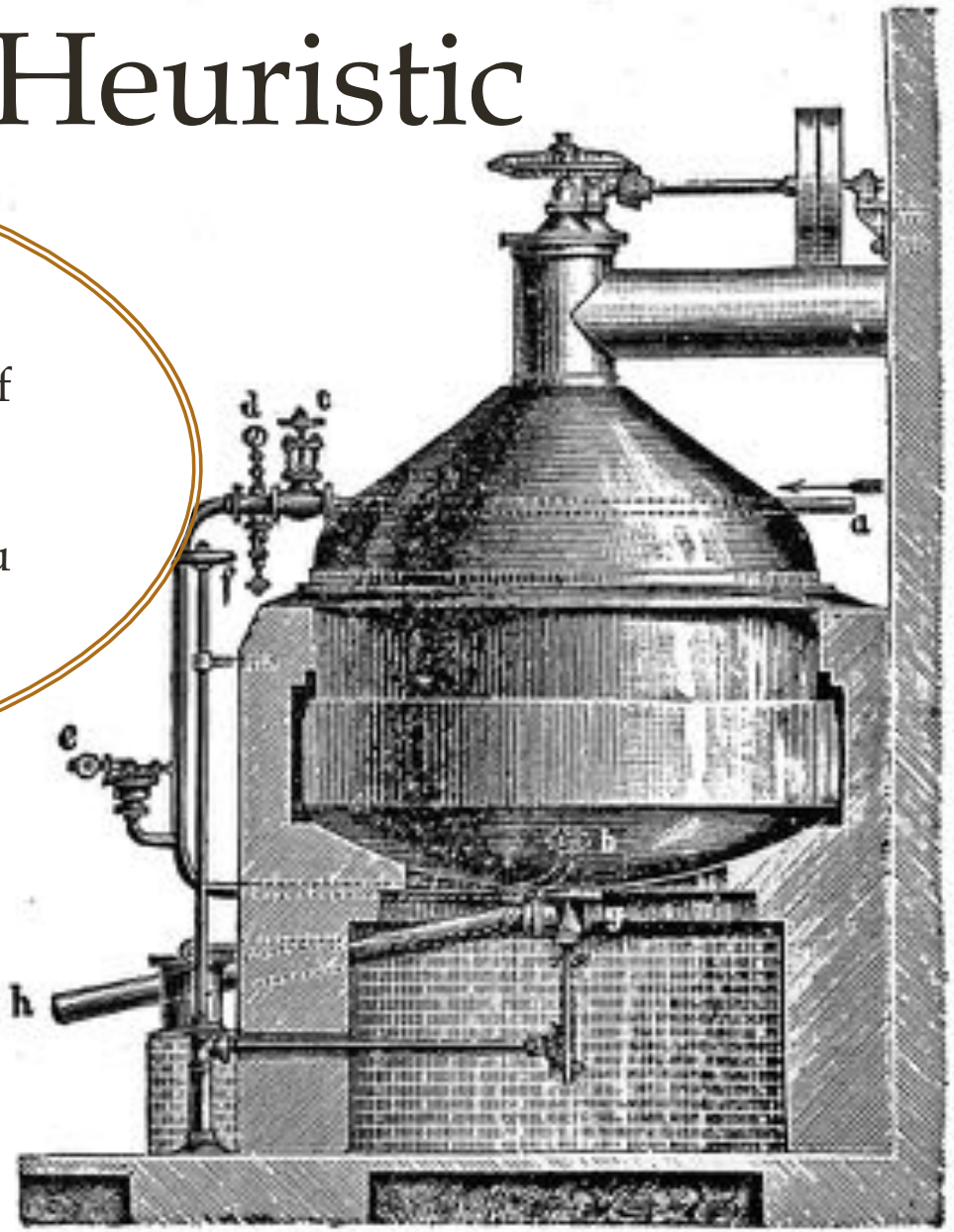
2. Take a view different from your preferred ways of working and argue both sides.

"As a rule, the more demanding the application, the more leverage you get from using a powerful language. But plenty of projects are not demanding at all. Most programming probably consists of writing little glue programs, and for little glue programs you can use any language that you're already familiar with and that has good libraries for whatever you need to do"

— Paul Graham, Revenge of the Nerds

Paul's Heuristic

It doesn't matter what programming language you use if you have a simple program. Use programming languages, tools, and frameworks and libraries you are familiar with.



My Imaginary Debate with Paul

But Paul, what about when something starts out simple but you know it will grow more complex over time?

And my lifelong heuristic:
Learn something new. Don't always do things the same way. That's soul sucking!

3. Have a conversation about a specific topic

My First Heuristics Distillation Conversation with Mathias Verraes

What's a heuristic you use when you model events?

***Heuristic*: Events are records of things that have happened, not things that will happen in the future.**

The event is "a reservation has been made" or "service has been scheduled"

Examples Keep the Conversation Flowing

Here's another heuristic: A bounded context should keep its internal details private.

Say if you keep monetary units with 10 digits precision internally in a service, pass out an amount with 2 digits precision because that's all other consumers of the event would need.

We Dig Deeper…

Perhaps there's another heuristic?

Design agreed upon standard formats based on standard usage.

Don't design message or event contents for specific subscribers to that event?

And then it got really interesting…

What happens if a new process needs extra precision?

Maybe it belongs within the bound context of the process that knows 10 digits precision?

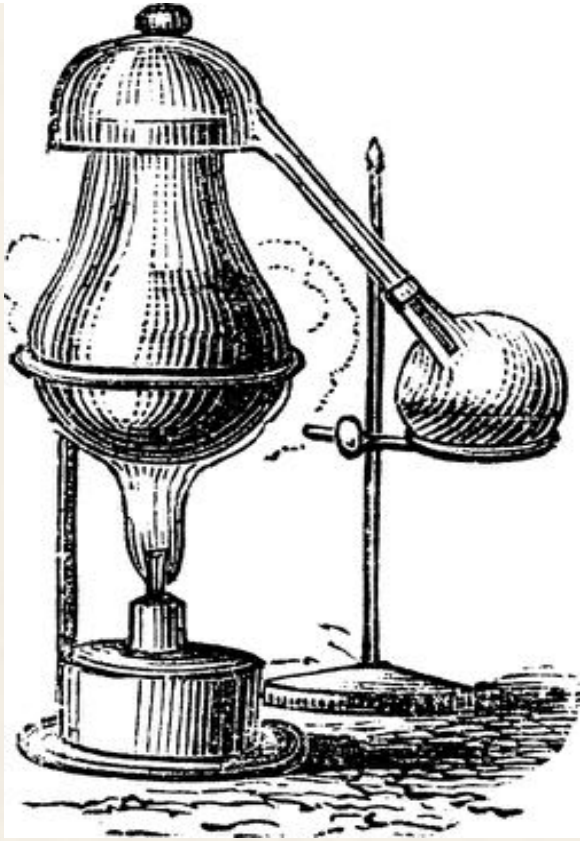
Which led to this insight…

These two heuristics compete!

Heuristic:
When designing information in an event, don't lose necessary precision.

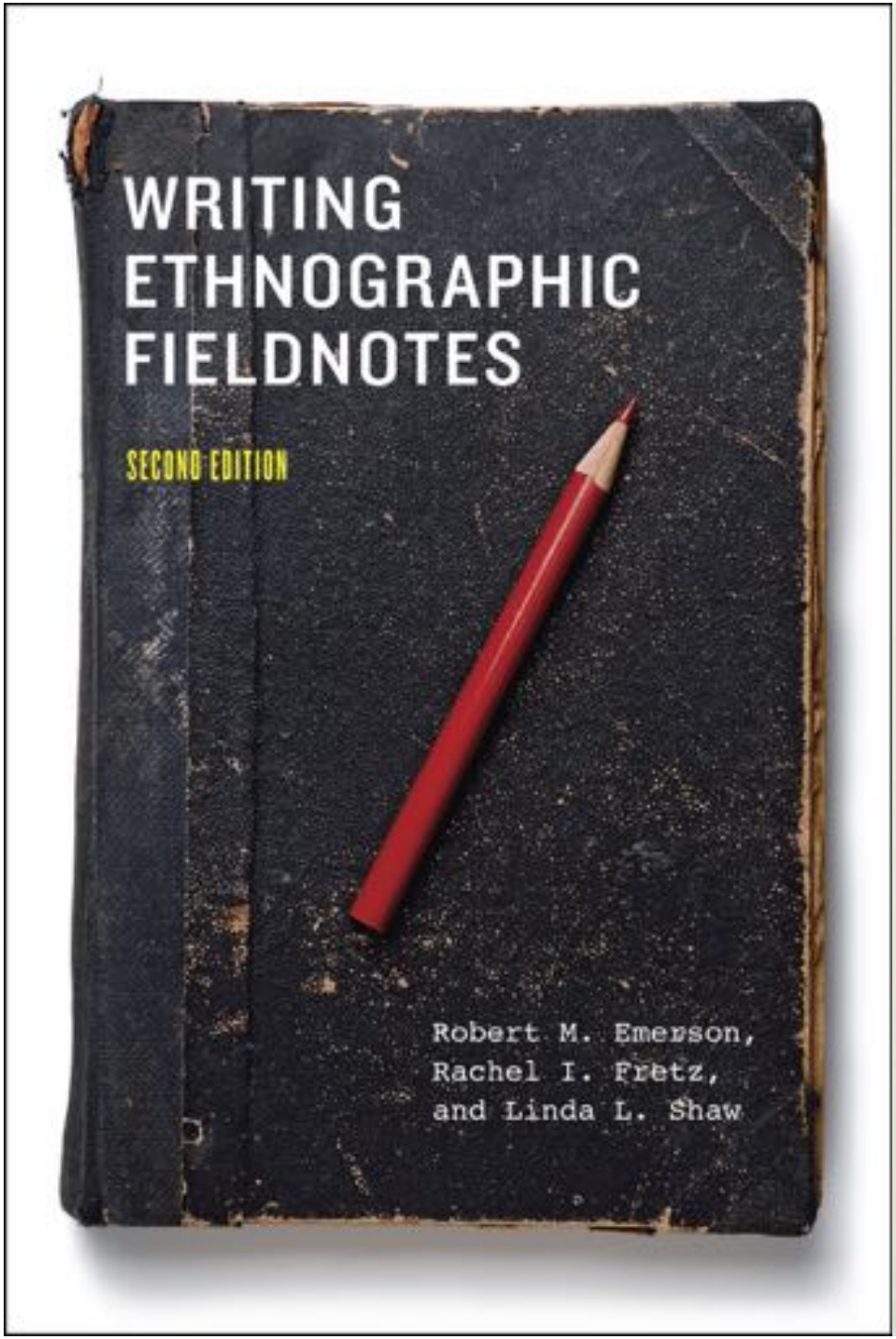
Heuristic:
Design agreed upon standard formats based on expected usage.

Distiller Advice



* Listen.
* Let the conversation flow.
* Ask questions:
  * Can you give me an example?
  * What happens if…?
* Do not record every heuristic in real time.
  * Photograph scribbles and drawings.
  * Record conversations for later.

4. *Radical Idea:* Take notes on how you work



When?

as you attempt
something new

you have a ½ hour

Notes from a Remote Mobbing Session with Cucumber folks

Third session's notes

On June 28th joined the mob when it was very small (just two). They were working on another enhancement: Adding the ability to approve a release. At one point it was just me with Julian…so I was really only observing/chipping in comments when there was a discussion.

Same time we were battling new pub/sub way of getting event info….

Heuristic: Who goes next as Driver is the one who has the most to learn (when rejoining the mob or perhaps when entering a new part of the system unfamiliar to them). Another way to say this: The stupidest person goes next.

Observation by Matt: There has been an ongoing tension between what is in the domain vs what is in the read model

Observation: There is also a tension between what the apis should be, what should be in projections….what information should be present, etc. The devil is in the details.

My observation: To make the smallest move possible while keeping (most of the tests passing), the team readily adds scaffolding code (e.g. keeping the code/API the same while making small, incremental changes to the code base.

Observation: Sometimes this extra scaffolding becomes more work than anticipated.

Heuristic: When there is something incomplete/not thorough about test code add @wip notation (ignored by CI tooling)--- examples @wip on incomplete assertions

Matt stated, we've learned not to name the tests for the event (there's a heuristic here about test names, I suspect)

We discussed about what "approval" means and when a release is "frozen" w/ respect to its tests. [I suspect that if "approval" is going to be an informal process, and that eventually multiple people might want to "approve" a release, then it might make sense to "freeze" a release and its tests at the time…or keep track of release approvals and the tests (that might be different?). This should not be overly complicated imo. Is the real req't auditability + visibility of collaborative state? Or???)]

Distill what you decide:

Document Design Decisions*

Title

Context - Forces at play

Decision - Stated with active voice: "We will ..."

Status - "proposed" or "accepted" later may be "deprecated" or "superseded"

Consequences positive, negative, and neutral that affect the team and project in the future

*Thanks to Michael Nygard
http://thinkrelevance.com/blog/2011/11/15/documenting-architecture-decisions
Useful link to github project on decision records: https://github.com/joelparkerhenderson

5. Distill What You Hear

Competing Event Source Evolution Heuristics

Michiel Overeem

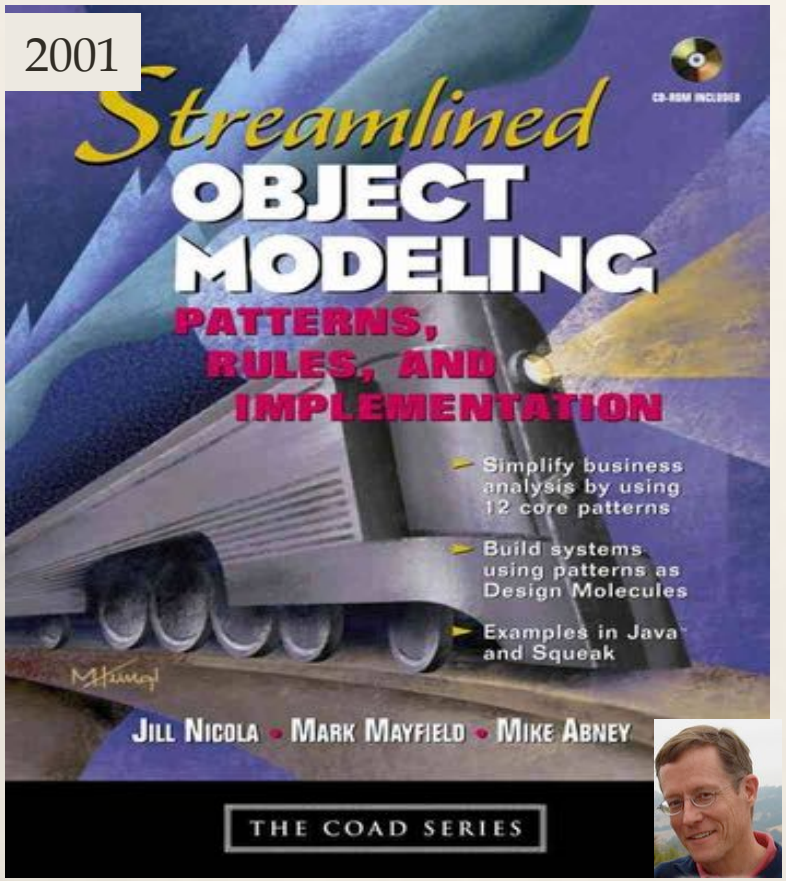

6. Revisit Your
Cherished Heuristics

Some of My Cherished Heuristics for Validating Data

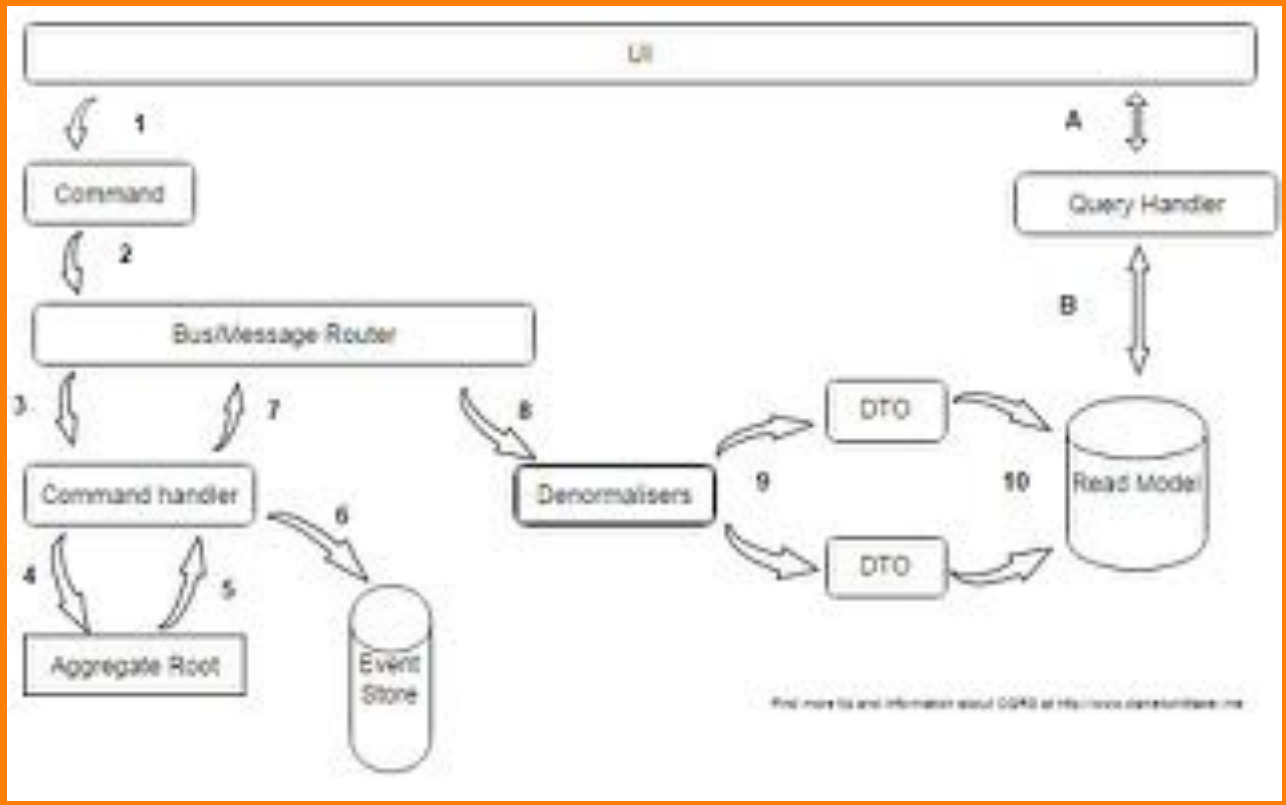
* Perform simple edits (syntactic) in browser code
* Don't trust browser-validated edits.
* Reapply them if receiving requests from an untrusted source
* Consistently assign validation responsibilities to framework-specific validation classes
* Consistently use domain validation and constraint enforcement patterns

One of My Heuristics: By characterizing a domain entity's attributes you can identify needed system behaviors

* *Descriptive Attributes* reflect a domain's properties (not identity).
* *Time-dependent attributes* Where maintaining a history of past values is important.
* *Lifecycle state attributes* Some entities go through a one-way lifecycle, from initial to final state.
* *Operational state* Some entities switch between different states. The state it is currently in determines how it behaves.



…what's different about validating/enforcing constraints within a CQRS architecture?



Heuristic*:
Distinguish between "superficial" and "domain" validations and handle them differently

1. "superficial": what must be true, regardless of the state of the domain

   Heuristic: Validate these before issuing a command, ideally on the client side as well as the server side

2. "superficial" but requires lookup of other information

   Heuristic: Validate in the service before invoking the command

3. "domain": validity of a command is dependent on the state of the model

   Heuristic: Validate in domain objects

*http://danielwhittaker.me/2016/04/20/how-to-validate-commands-in-a-cqrs-application/

Let's Just Get On With It

Sorting things out…

superficial vs. domain validations

syntactic vs. semantic validations

descriptive attributes vs.
time-dependent attributes vs.
life cycle attributes vs.
operational state attributes

location? constraints?

…notice what happens

when you apply a heuristic

when you back up and try something else

when you disagree on what to do next

Keep Your Heuristics Alive

- Practice being intentional
- Write and share your heuristics with others
- Expect them to grow and evolve

@rebeccawb

Thank you!

rebecca@wirfs-brock.com
www.wirfs-brock.com/blog
www.wirfs-brock.com

Heuristics blog posts:

wirfs-brock.com/blog/2019/03/20/growing-your-personal-design-heuristics

wirfs-brock.com/blog/2019/04/04/what-do-typical-design-heuristics-look

wirfs-brock.com/blog/2019/04/12/writing

wirfs-brock.com/blog/2019/04/19/nothing-ever-goes-exactly-by-the-book

wirfs-brock.com/blog/2019/04/25/what-we-say-versus-what-we-do

DDD Heuristics website:

https://www.dddheuristics.com

@rebeccawb