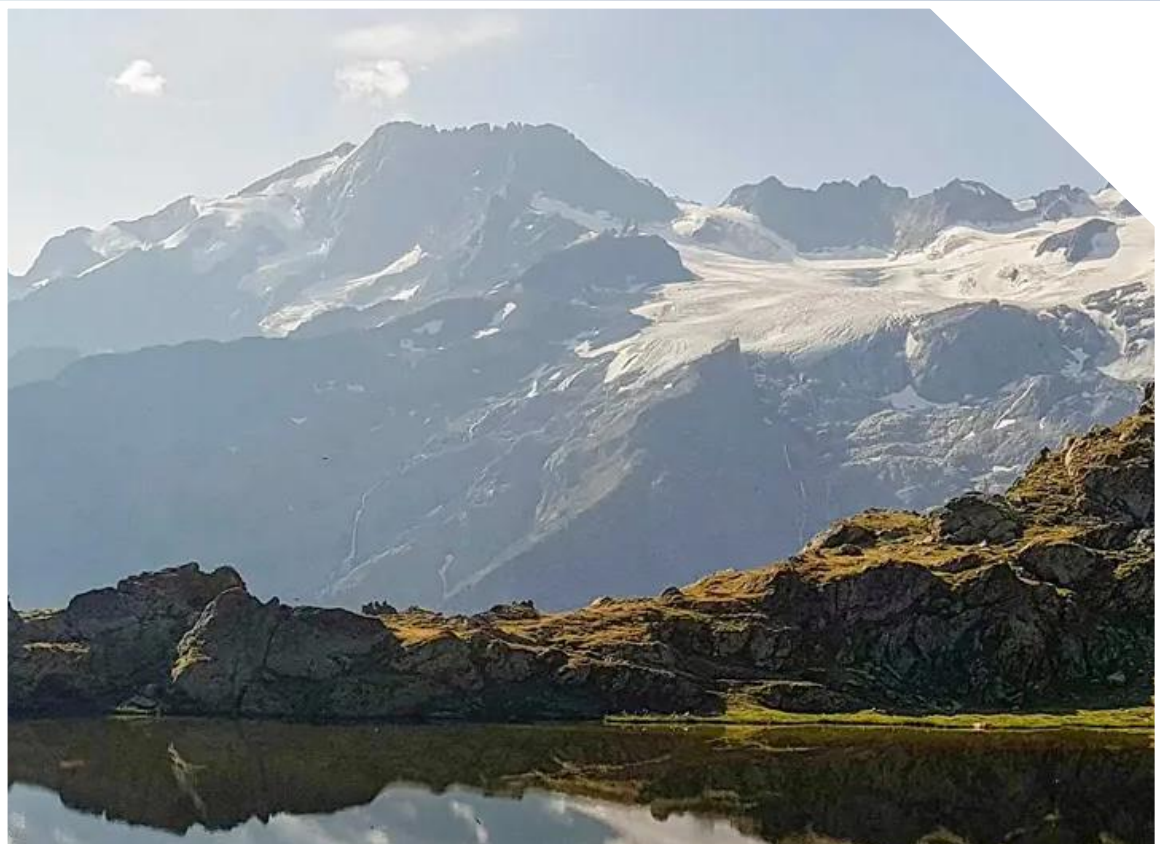


TIDY FIRST ?

PRACTICAL GUIDE FOR SUSTAINABLE AND COST-EFFECTIVE CODE

ARNAUD THIÉFAINE

JUNE 2024

ONE YEAR AGO …

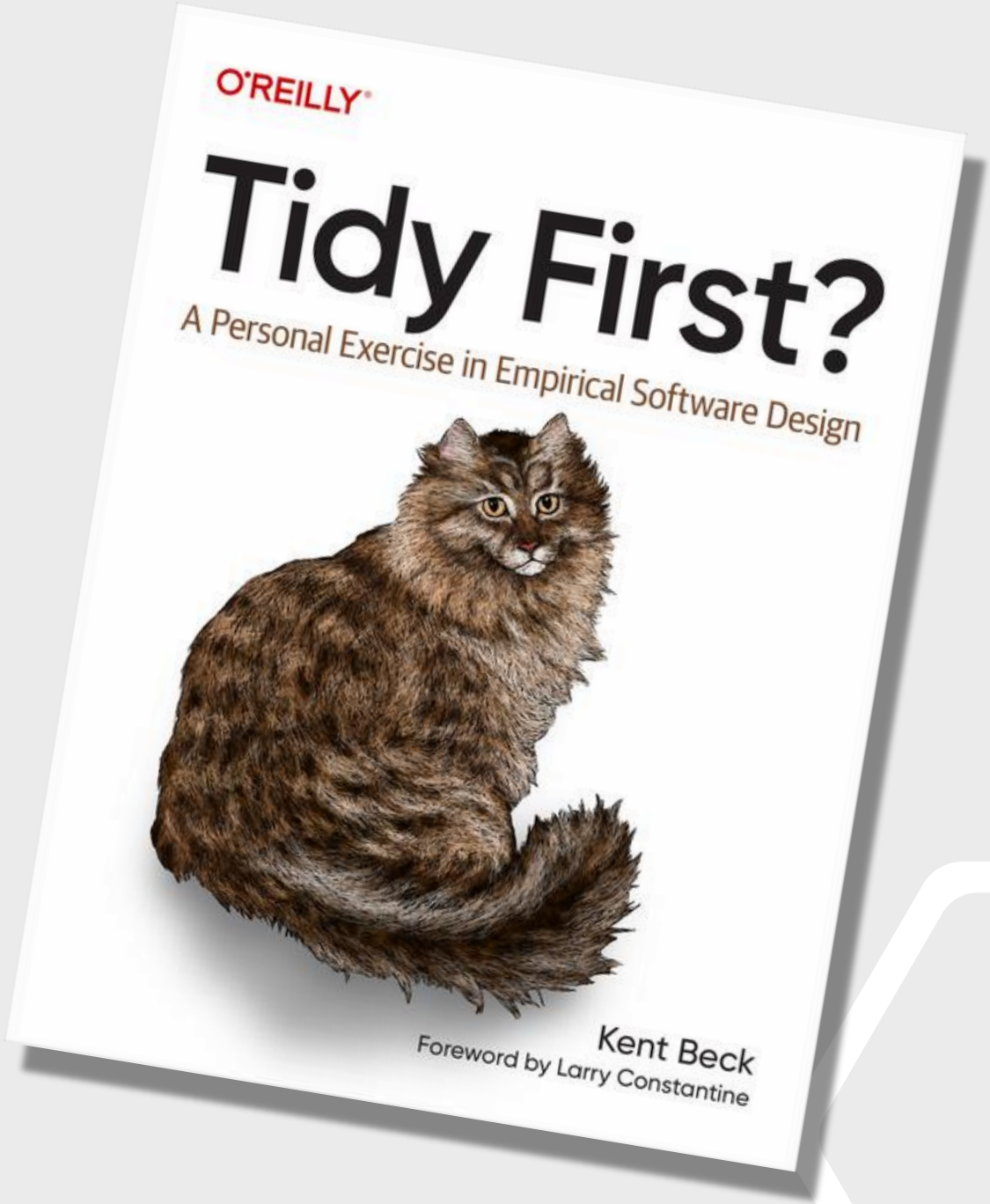

DOMAIN
DRIVEN
DESIGN
EUROPE
arolla
DEVS, LET'S RECLAIM DDD!
Arnaud THIEFAINE
Dorra BARTAGUIZ

ONE YEAR AGO …

DO YOU KNOW HIM ?

KENT BECK


The Addison Wesley Signature Series
Test-Driven Development
By Example
Kent Beck
A Kent Beck Signature Book
The Addison Wesley Signature Series
Implementation Patterns
Kent Beck
A Kent Beck Signature Book
Quick Look-up and Advice
JUnit
Pocket Guide
O'Reilly
Kent Beck
Copyrighted Material
Extreme Programming Explained
Embrace Change
Kent Beck
with Cynthia Andres
Foreword by Erich Gamma
Second Edition


O'REILLY
Tidy First?
A Personal Exercise in Empirical Software Design
Kent Beck
Foreword by Larry Constantine

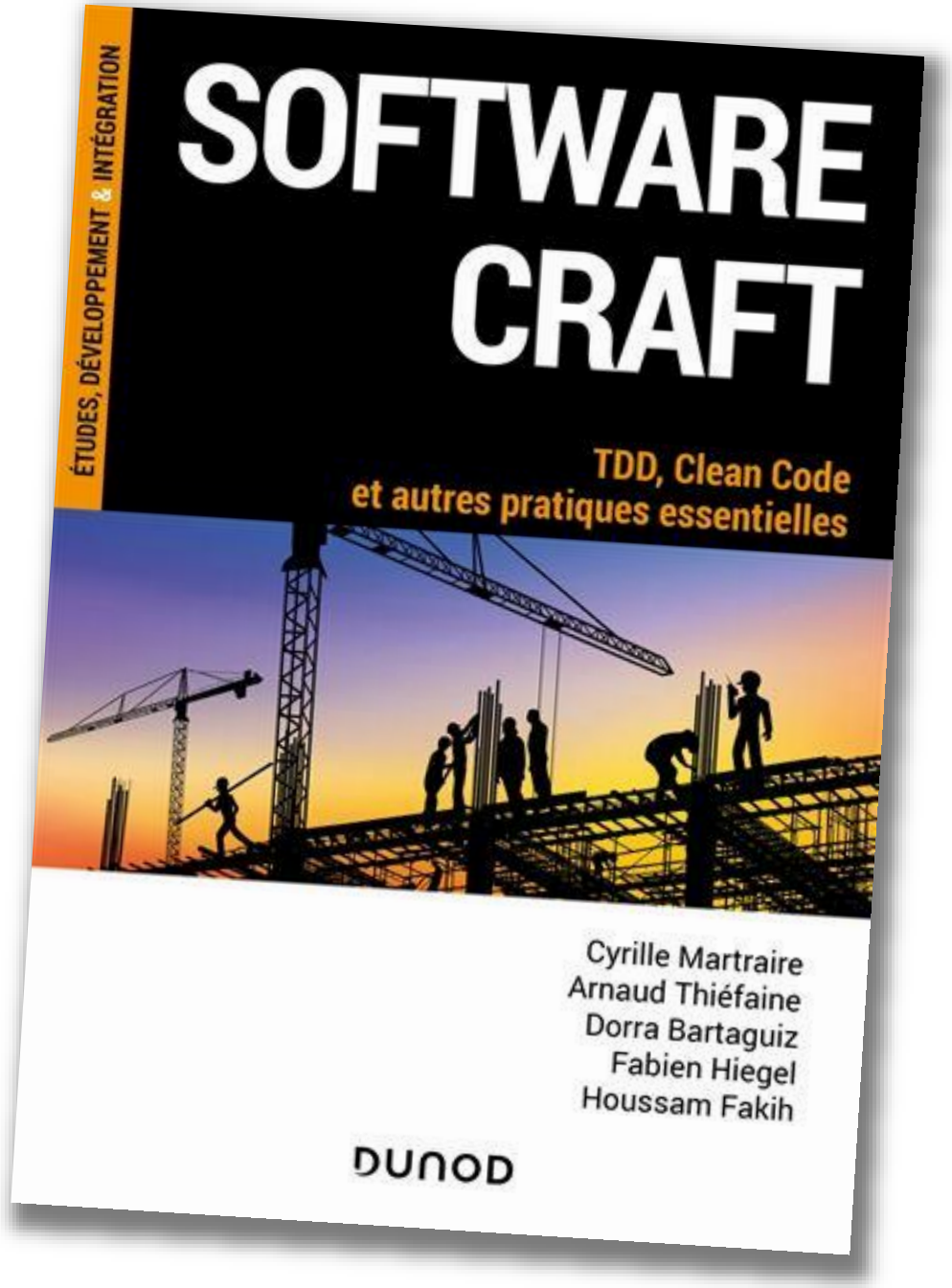
ABOUT ME

- xx years of experience
- Crafter for many years
- Author

@arnaudthiefaine

athiefaine

arnaudthiefaine



KAIZEN SOLUTIONS

APPLIED MATERIALS®

ATEME
Captivate your audience

CBTW

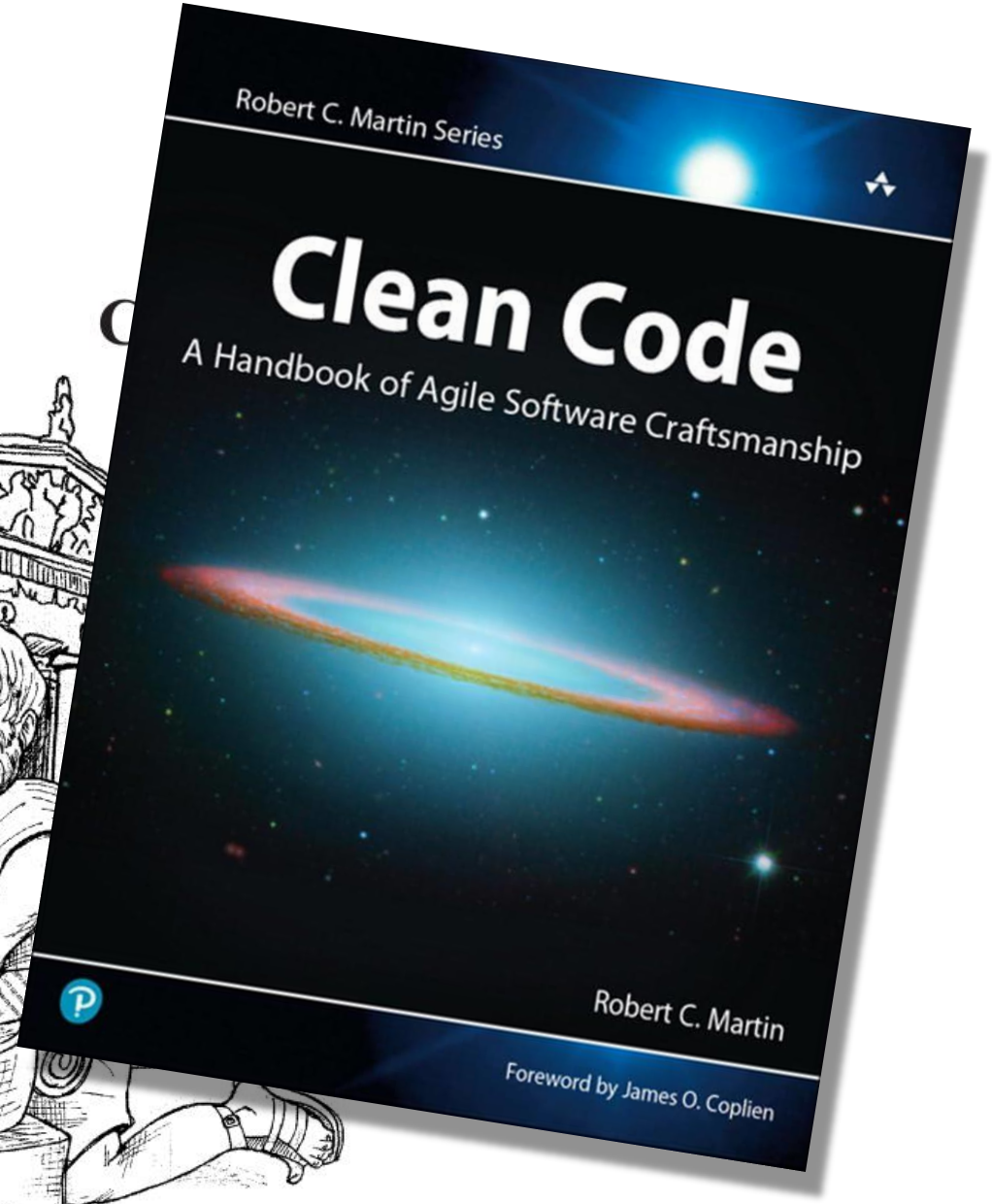
Julien Lenormand
#human #craft

tuleap

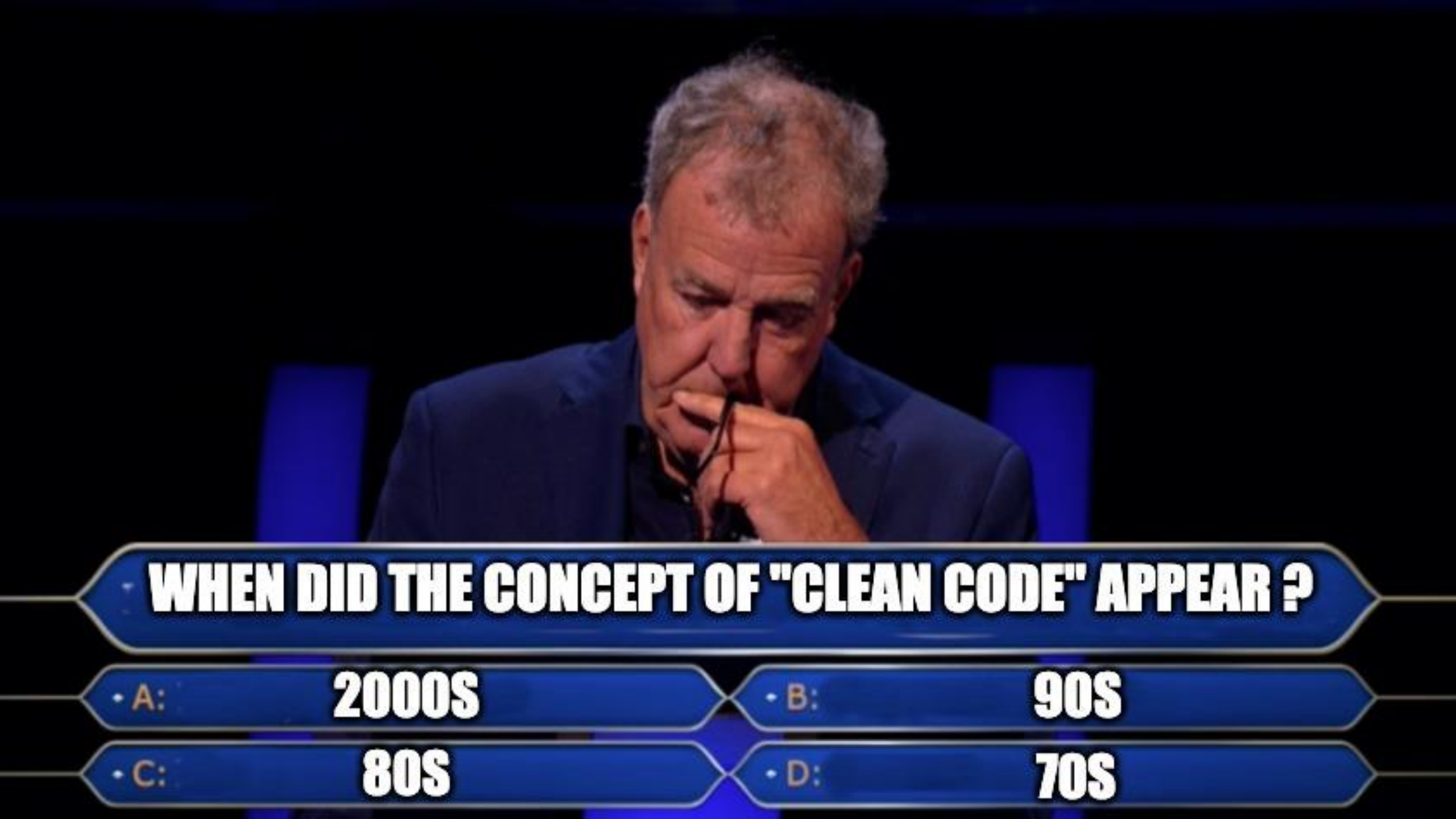
zenika

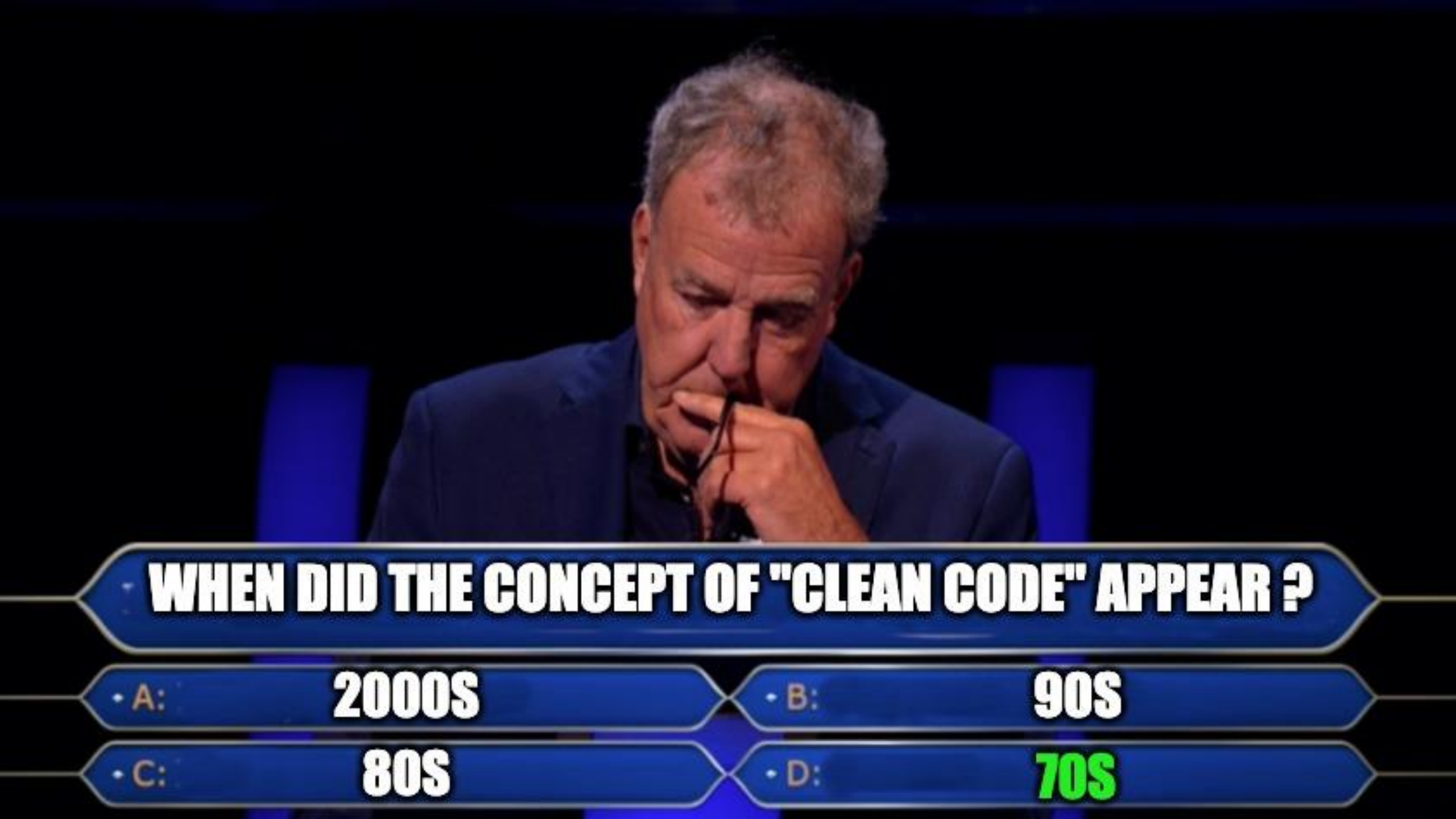

WHAT IS TIDYING ?

WHAT IS TIDYING ?



(from *Clean Code* book by Robert C. Martin)


WHEN DID THE CONCEPT OF "CLEAN CODE" APPEAR ?
A: 2000S
B: 90S
C: 80S
D: 70S


WHEN DID THE CONCEPT OF "CLEAN CODE" APPEAR ?
A: 2000S
B: 90S
C: 80S
D: 70S

1968

Edgar Dijkstra: Go To Statement Considered Harmful

```
C     PROGRAM TO COMPUTE PRIME NUMBE
C
      PROGRAM PRIME
      INTEGER MAXINT, N, DIVSOR
      INTEGER QUOT, PROD
      READ(*,100) MAXINT
      N=2
      WRITE(*,150) MAXINT
5     IF (MAXINT-N) 200,10,10
10    DIVSOR = 2
15    IF ((N-1)-DIVSOR) 30,20,20
20    QUOT = INT(N/DIVSOR)
      PROD = INT(QUOT*DIVSOR)
      IF (N-PROD) 25,30,25
25    DIVSOR = DIVSOR + 1
      GO TO 15
30    IF (DIVSOR-(N-1)) 40,40,35
35    WRITE (*,100) N
40    N=N+1
      GO TO 5
100   FORMAT(I5)
150   FORMAT('THE PRIME NUMBERS FROM 2 TO ',I5,' ARE: ')
200   STOP
      END
```

Go To Statement Considered Harmful

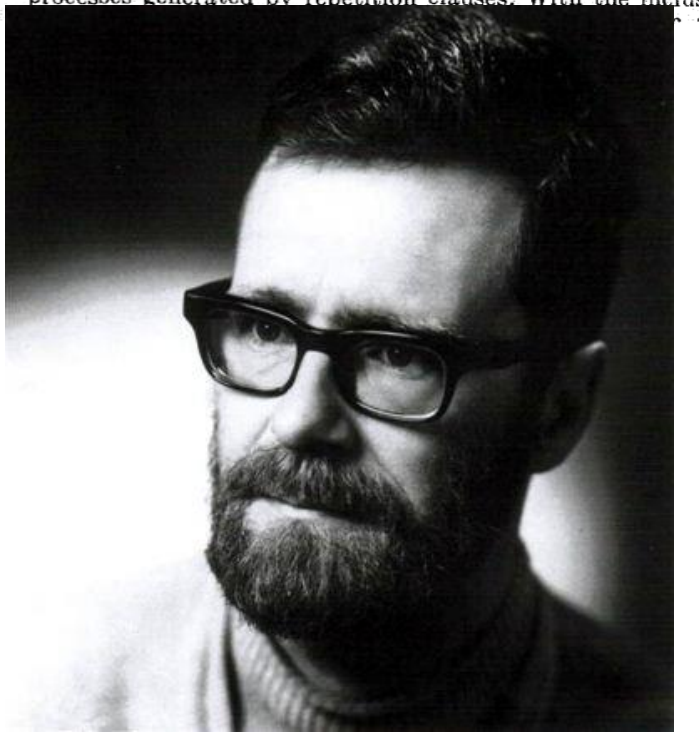
Key Words and Phrases: go to statement, jump instruction, branch instruction, conditional clause, alternative clause, repetitive clause, program intelligibility, program sequencing
CR Categories: 4.22, 5.23, 5.24

EDITOR:

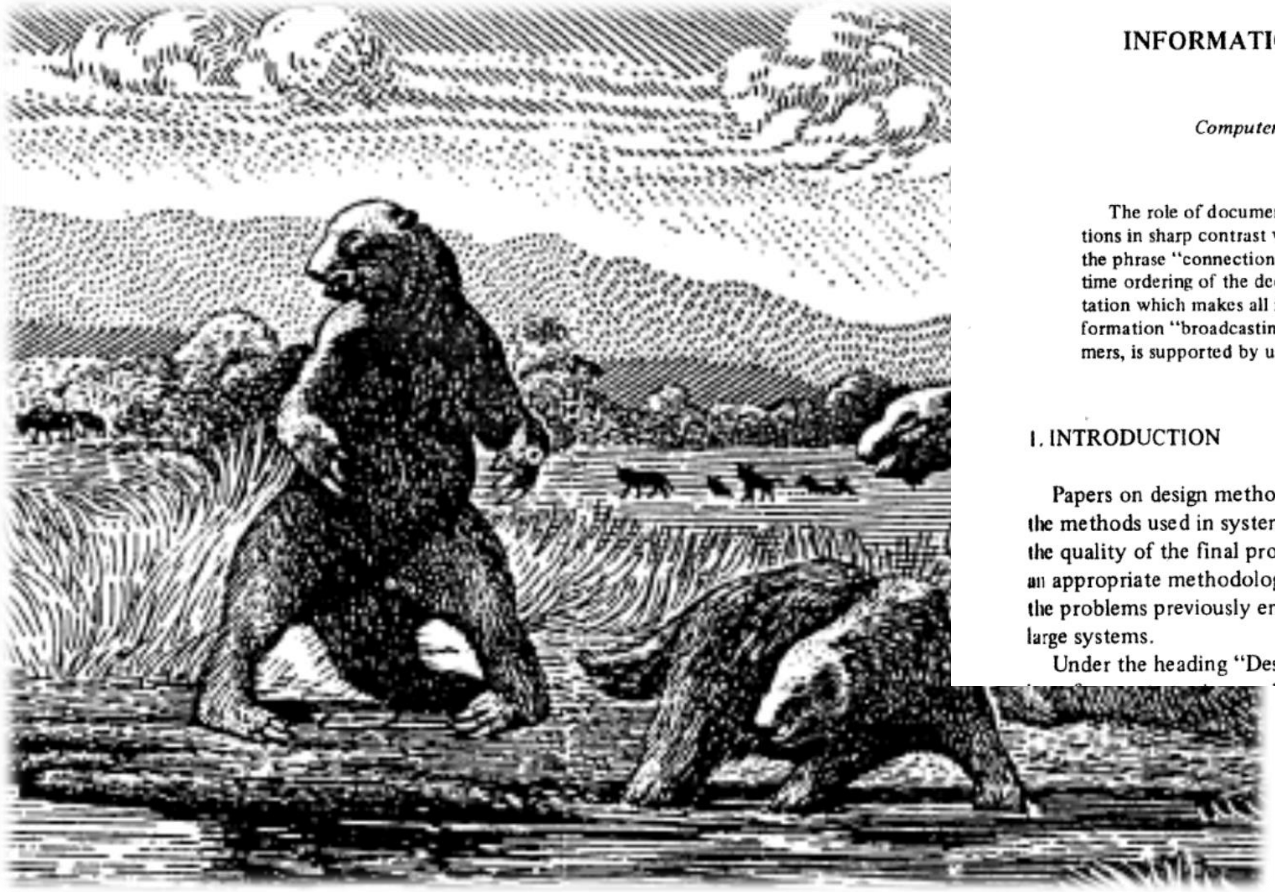
For a number of years I have been familiar with the observation that the quality of programmers is a decreasing function of the density of **go to** statements in the programs they produce. More recently I discovered why the use of the **go to** statement has such disastrous effects, and I became convinced that the **go to** statement should be abolished from all "higher level" programming languages (i.e. everything except, perhaps, plain machine code). At that time I did not attach too much importance to this di-

dynamic progress is only characterized when we also give to which call of the procedure we refer. With the inclusion of procedures we can characterize the progress of the process via a sequence of textual indices, the length of this sequence being equal to the dynamic depth of procedure calling.

Let us now consider repetition clauses (like, **while** *B* **repeat** *A* or **repeat** *A* **until** *B*). Logically speaking, such clauses are now superfluous, because we can express repetition with the aid of recursive procedures. For reasons of realism I don't wish to exclude them: on the one hand, repetition clauses can be implemented quite comfortably with present day finite equipment; on the other hand, the reasoning pattern known as "induction" makes us well equipped to retain our intellectual grasp on the processes generated by repetition clauses. With the inclusion of



1971



(from *The Mythical Man Month* by Fred Brooks)

INFORMATION PROCESSING 71 – NORTH-HOLLAND PUBLISHING COMPANY (1972)

INFORMATION DISTRIBUTION ASPECTS OF DESIGN METHODOLOGY *

D.L. PARNAS

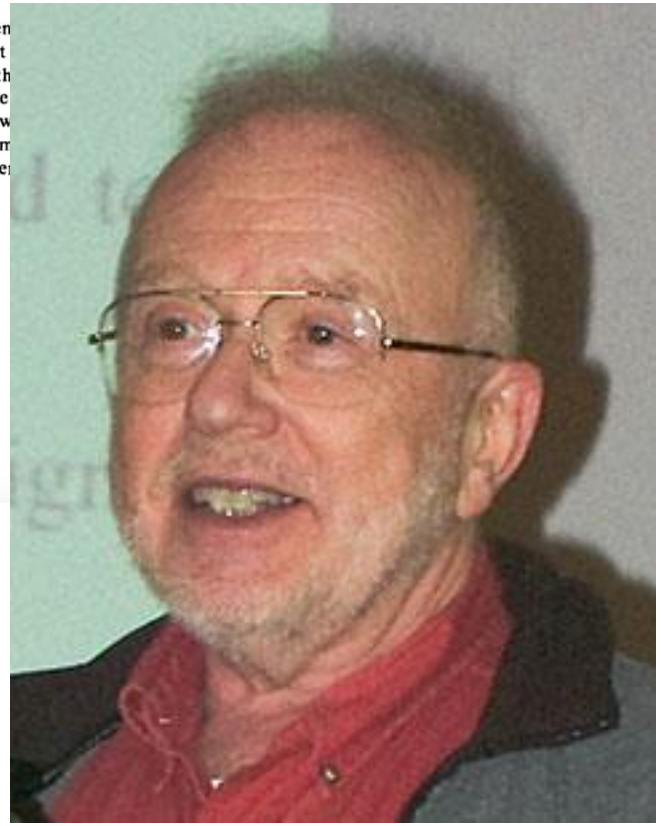
Computer Science Department, Carnegie-Mellon University, Pittsburgh, Pennsylvania, USA

The role of documentation in the design and implemen
tions in sharp contrast with current practice. The concept
the phrase "connections between modules". It is shown th
time ordering of the decisions) may be inconsistent. Some
tation which makes all information accessible to anyone w
formation "broadcasting" is harmful, that it is helpful if m
mers, is supported by use of the above mentioned conside

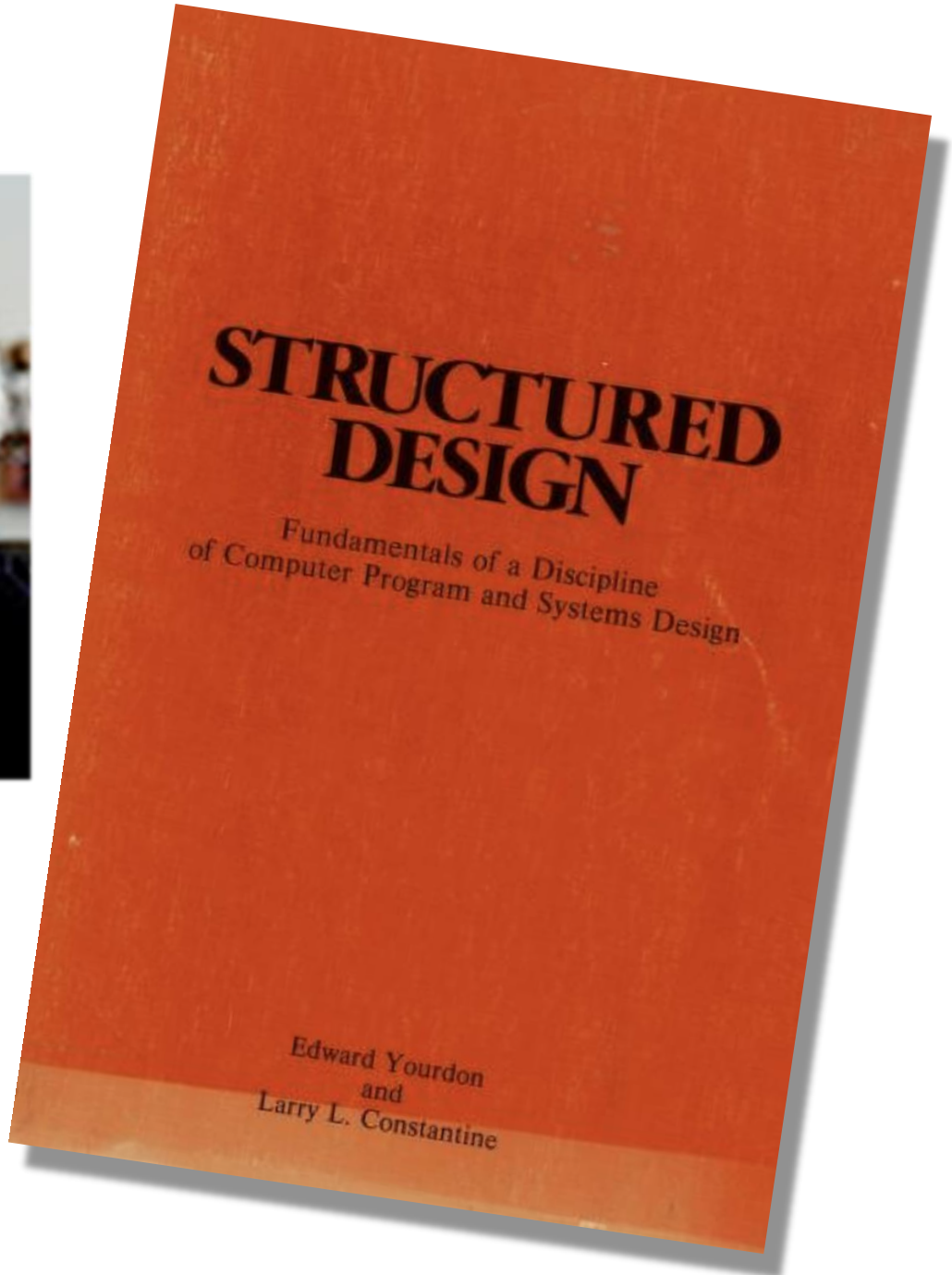
1. INTRODUCTION

Papers on design methodology assume (1) that the methods used in system design strongly affect the quality of the final product; and (2) by selecting an appropriate methodology we can avoid many of the problems previously encountered in constructing large systems.

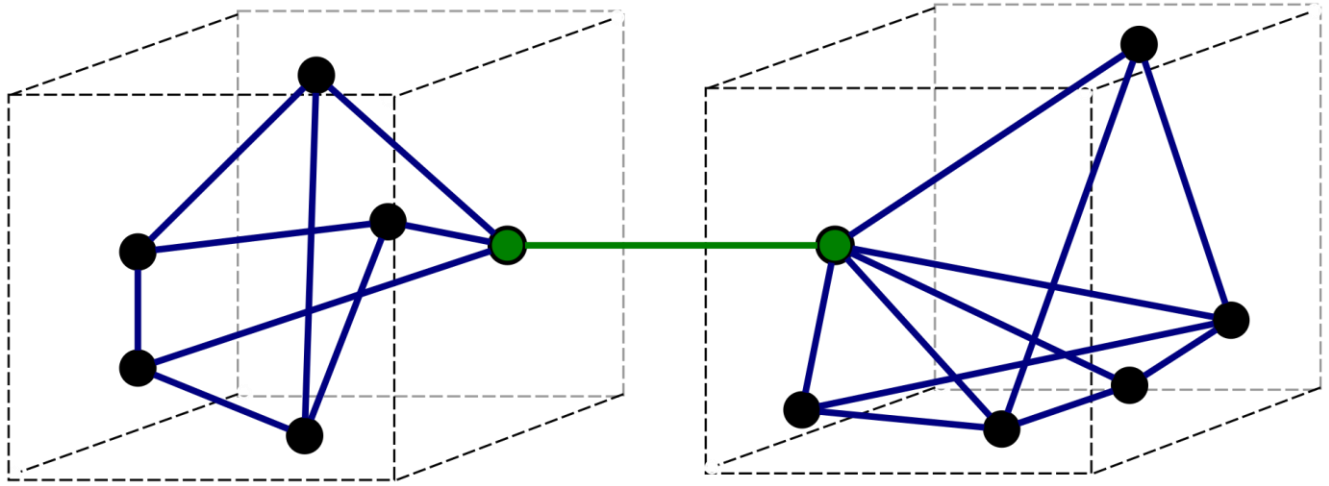
Under the heading "Design Methodology" a num-



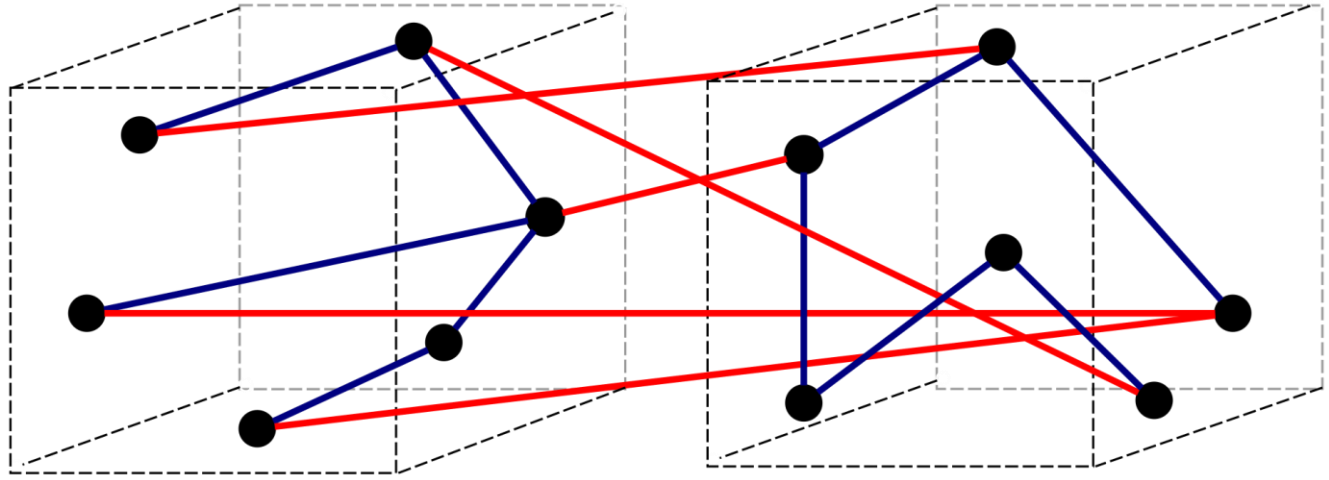
1974


STRUCTURED
DESIGN
Fundamentals of a Discipline
of Computer Program and Systems Design
Edward Yourdon
and
Larry L. Constantine

COUPLING VS COHESION



a) Good (loose coupling, high cohesion)



b) Bad (high coupling, low cohesion)

https://en.wikipedia.org/wiki/Coupling_%28computer_programming%29#/media/File:CouplingVsCohesion.svg

A LONG TRADITION OF CLEAN CODE

TIDYING VALUE

boy scout rule
as a mindset

baby steps

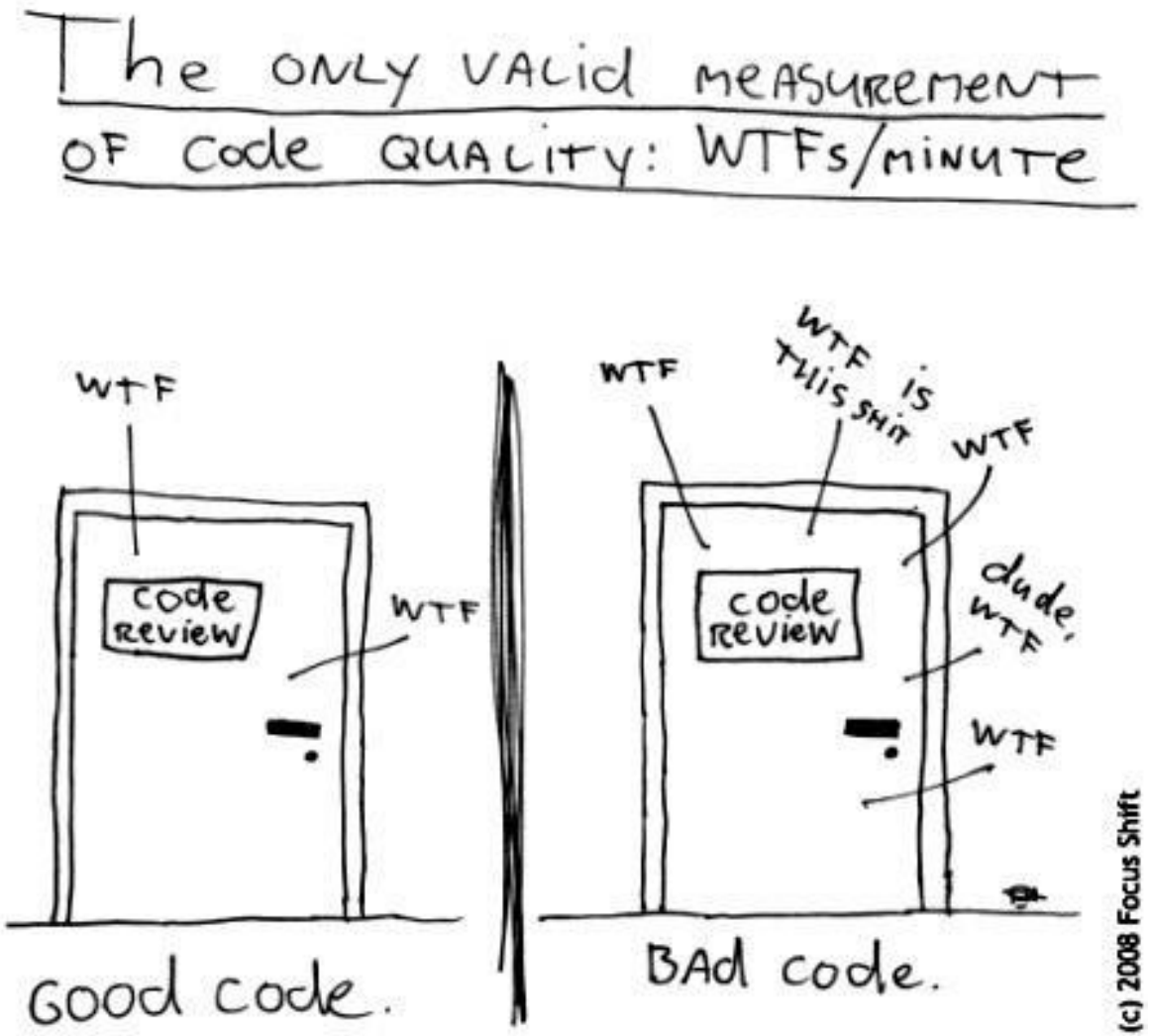
micro-refactoring

low-priced

easy to learn
easy to master
easy to apply

code hygiene enforcer

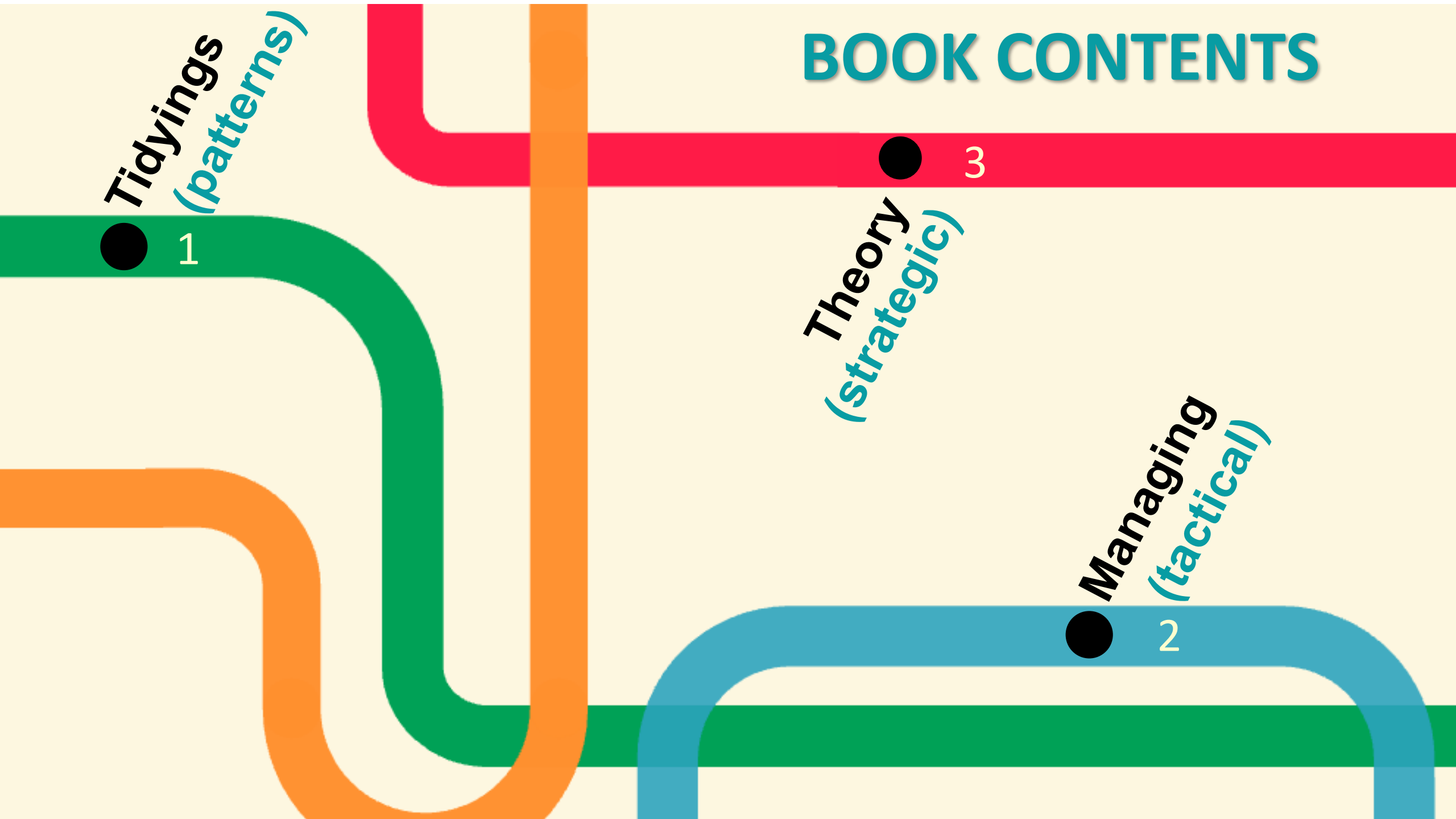
WHY DO WE NEED TIDYING ?



“The biggest cost of code is the cost of reading and understanding it, not the cost of writing it.”

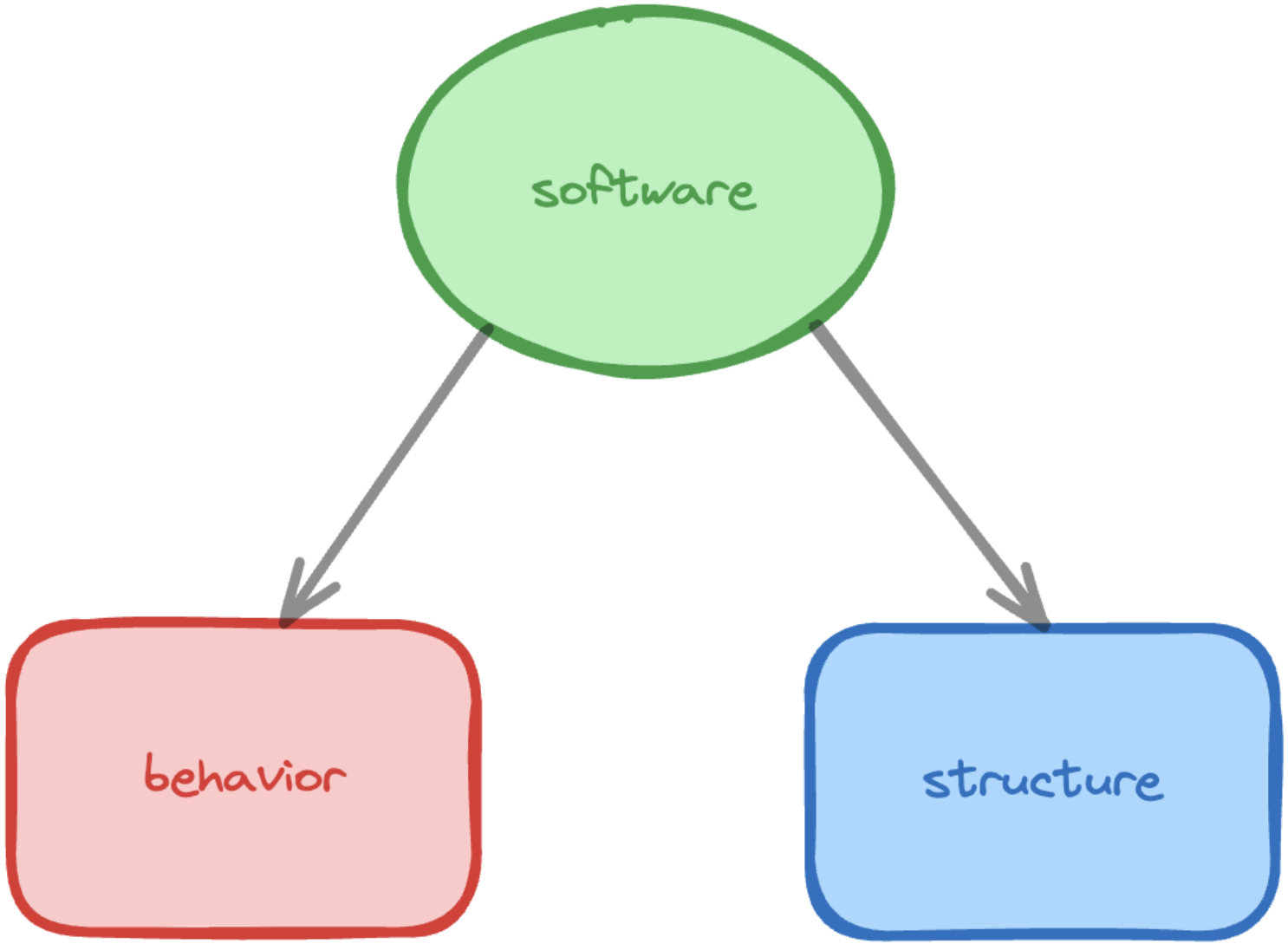
(from *Tidy First ?* By Kent Beck)

BOOK CONTENTS

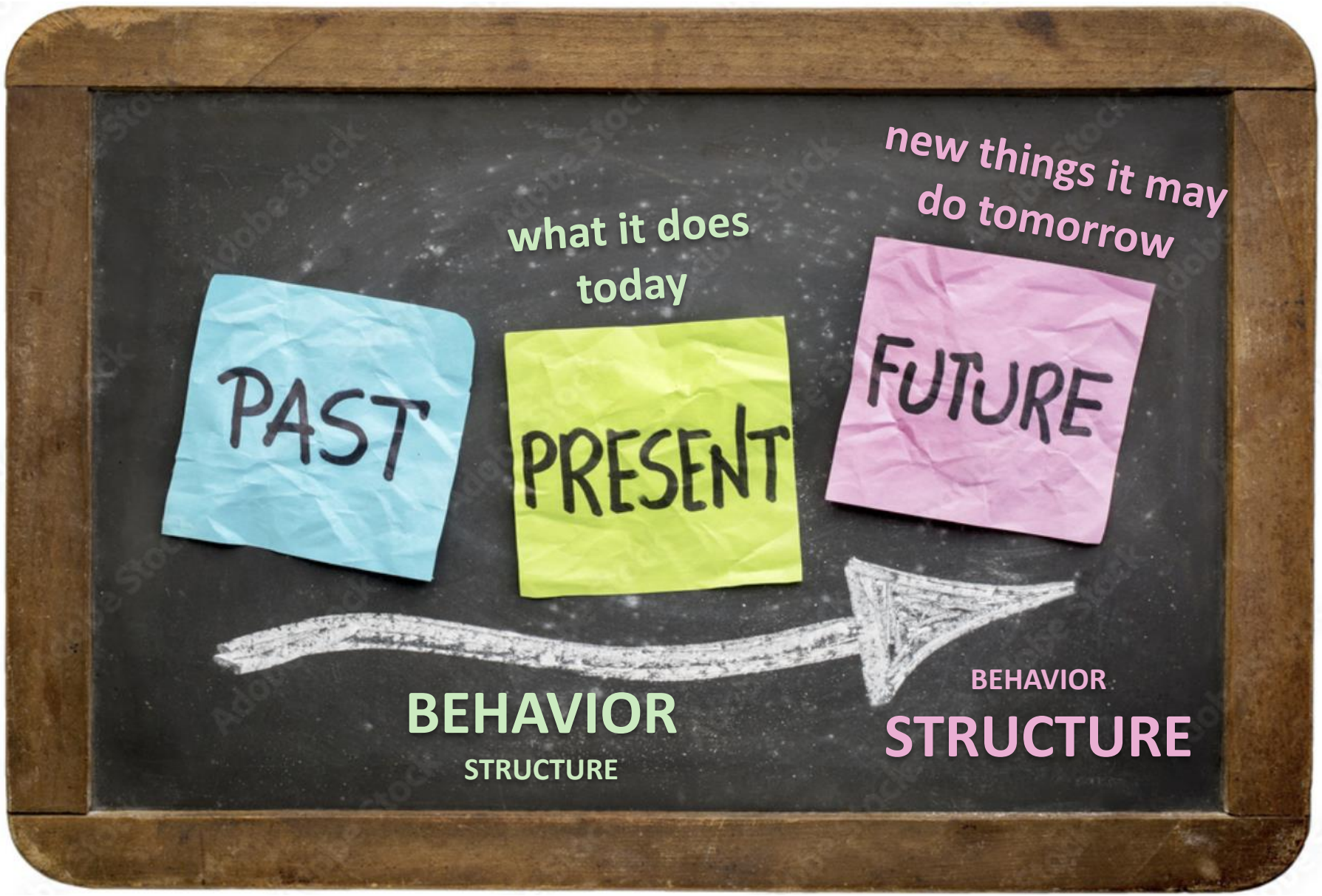



STRATEGIC
TIDYING

VALUE OF SOFTWARE


software
behavior
structure

VALUE OF SOFTWARE

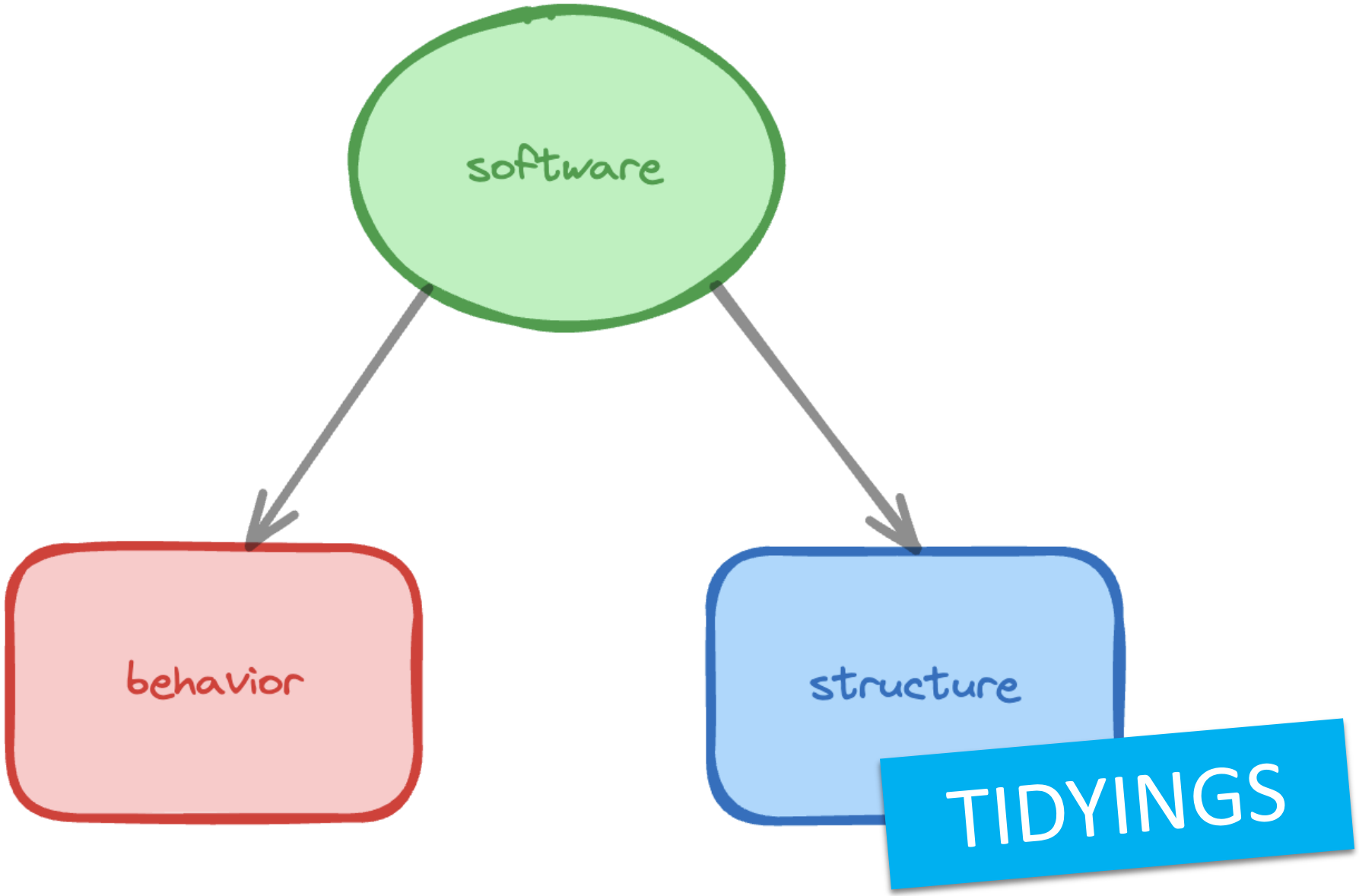



FLEXIBILITY
MODULARITY

REVERSIBILITY

hard-learned boy

scout rule


software
behavior
structure
TIDYINGS

TIME VALUE OF MONEY

1€ today > 1€ tomorrow

OPTIONS

underlying (with price)

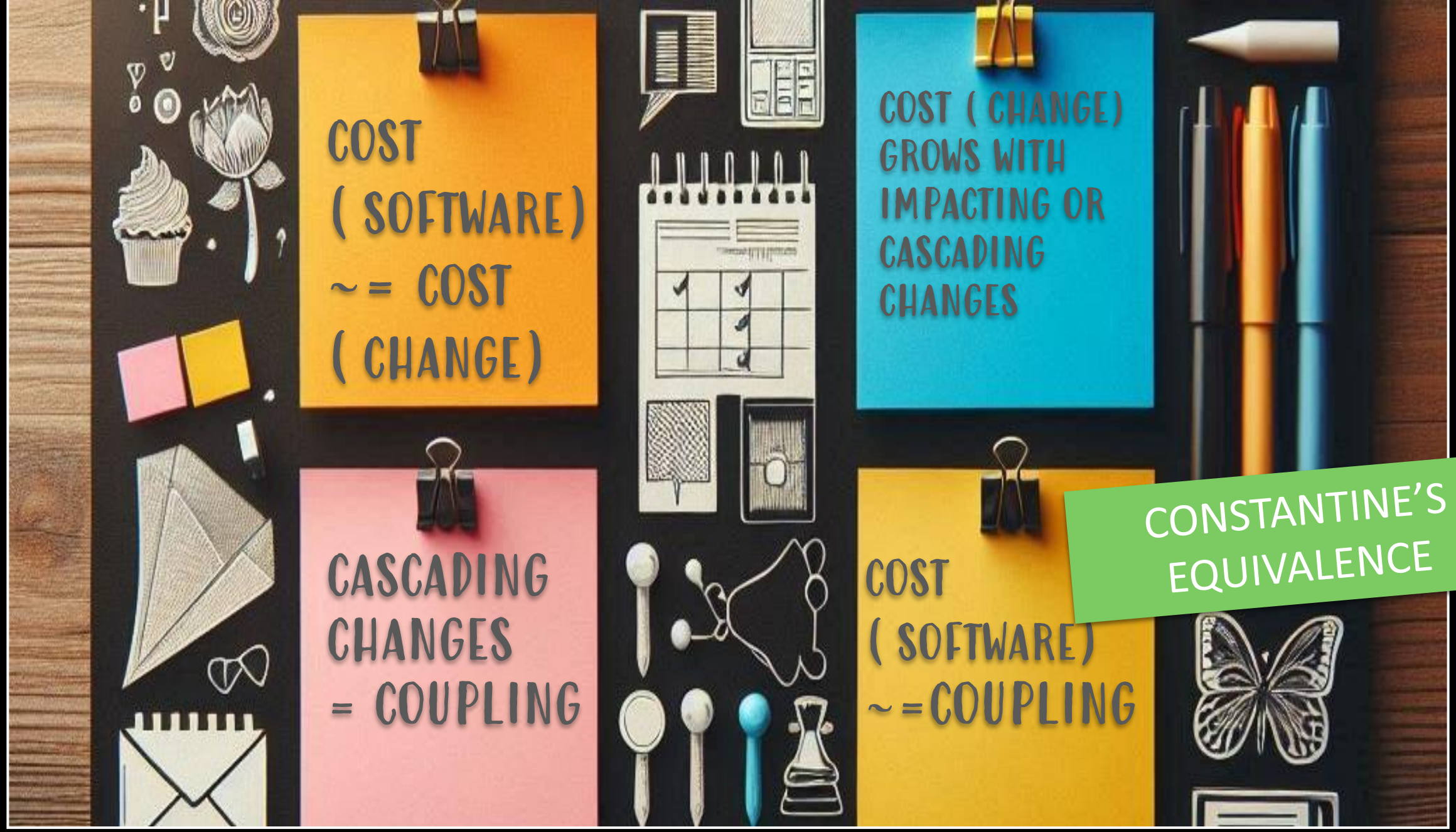
option premium

OPTIONALITY IN SOFTWARE

software design = preparation for change (of behavior)

behavior change (tomorrow) = option underlying

structure change (today) = option premium

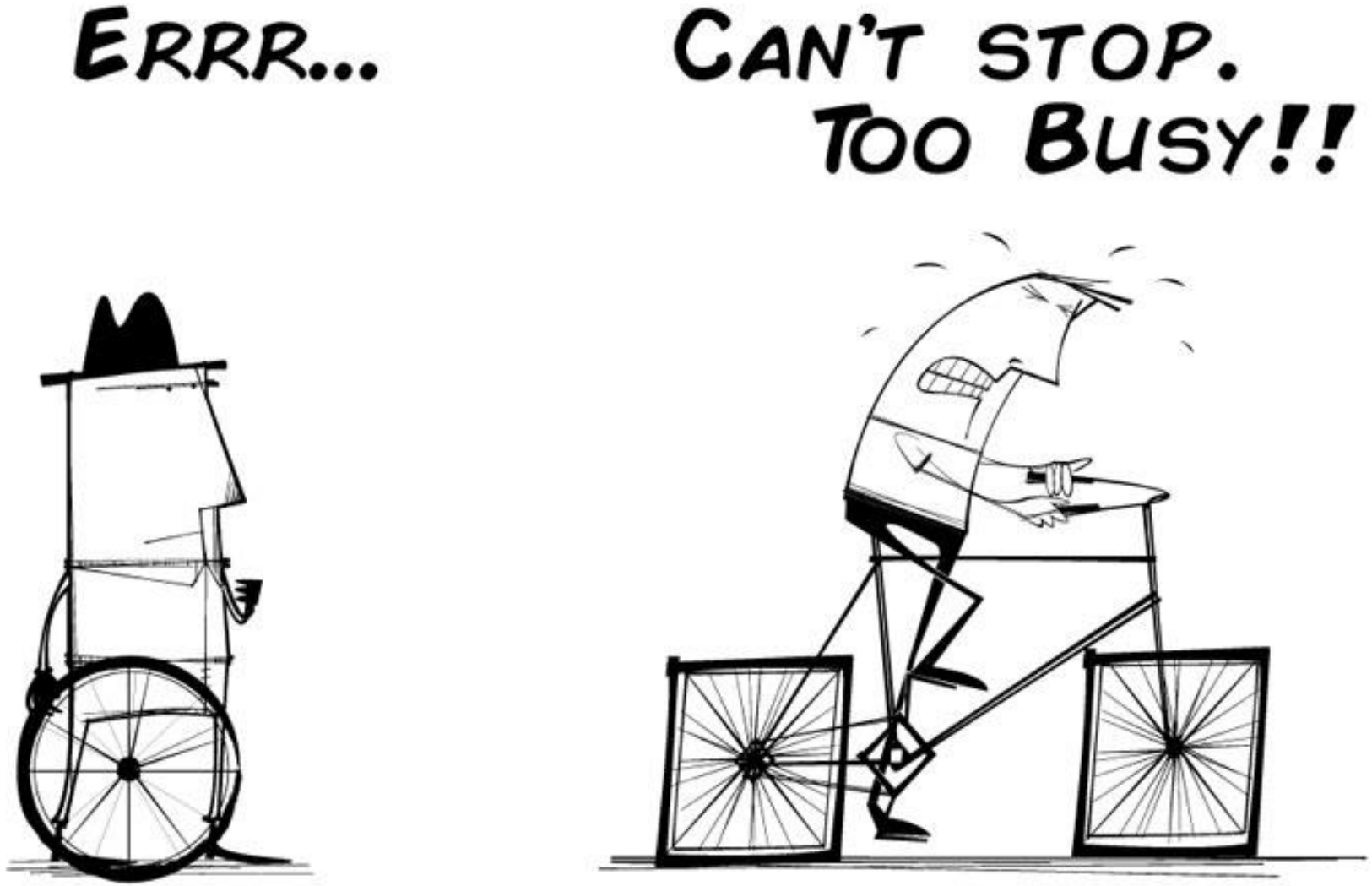


COST OF SOFTWARE

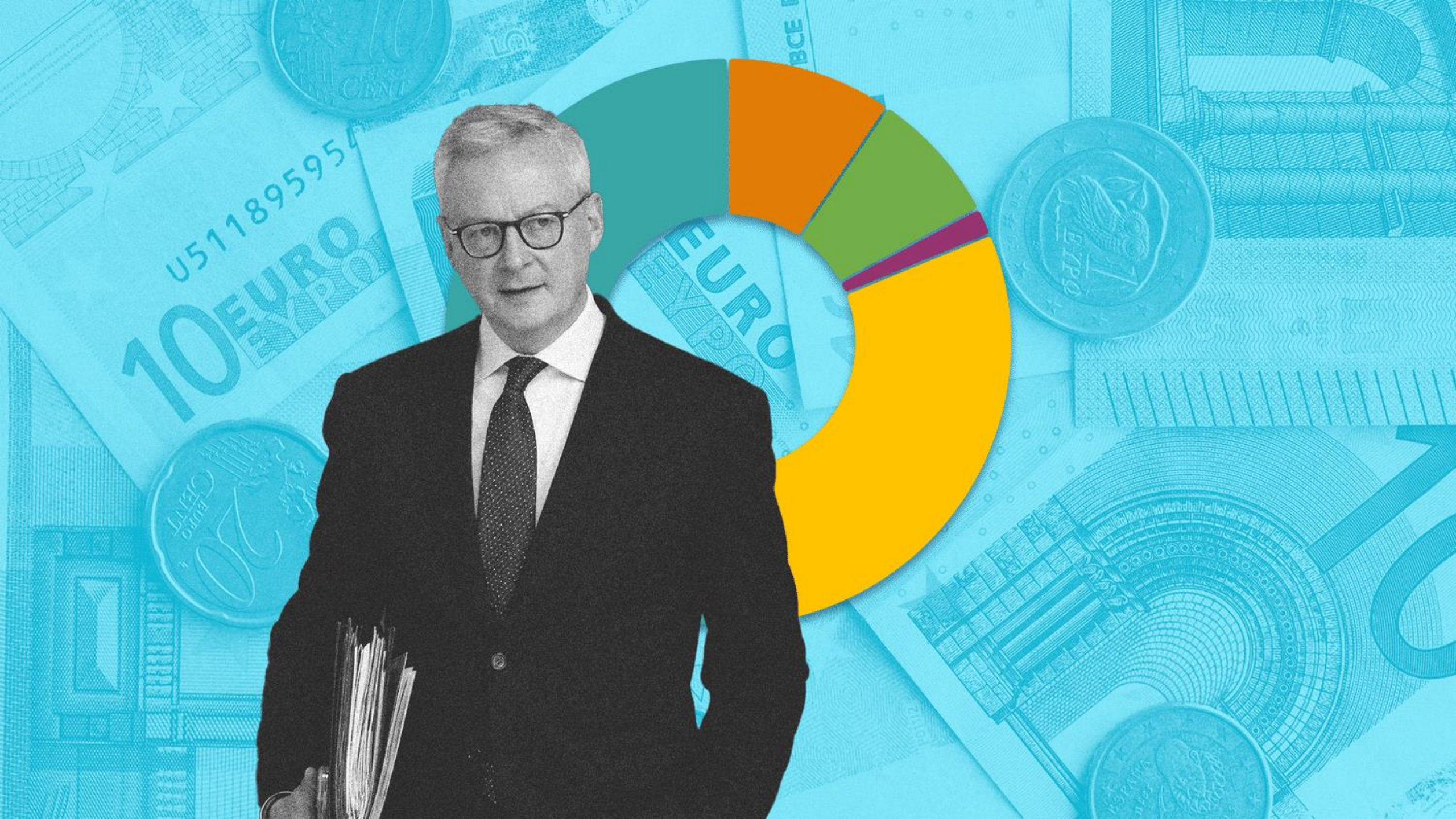
COUPLING VS DECOUPLING

excruciating pain !!!

TECHNICAL DEBT

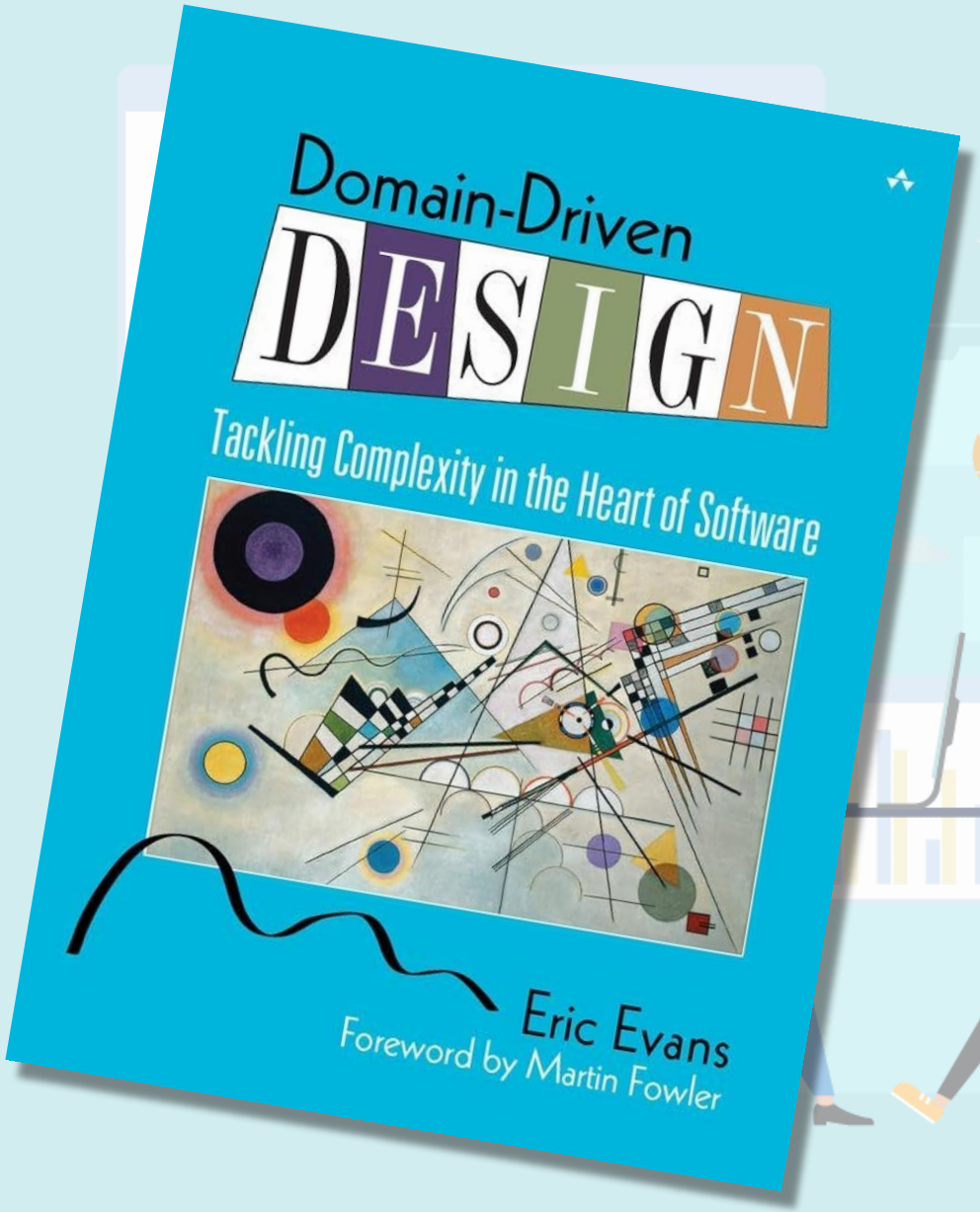


https://jamesreynolds.github.io/devops/2019/02/14/an-empirical-measurement-of-technical-debt.html


U51189595
10 EURO
EUROPA
EURO


IMPROVING
OPTIONALITY

CHOOSING THE RIGHT UNDERLYING




TACTICAL
TIDYING


TIDYING
PATTERNS

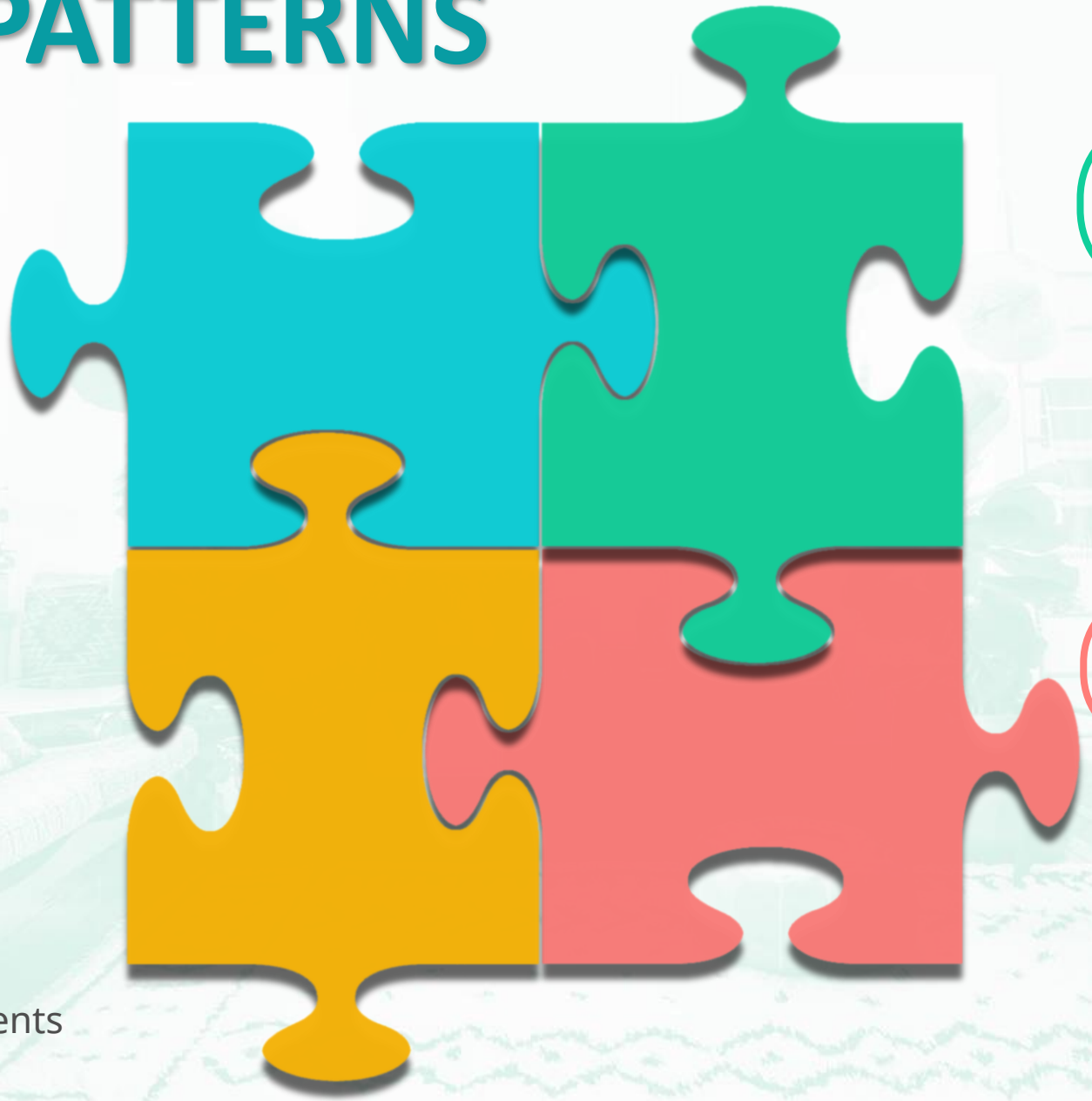
TIDYING PATTERNS

PRESENTATION AND READABILITY

- Chunk statements
- Normalize symmetries
- Dead code
- Reading order

KNOWLEDGE AND DOCUMENTATION

- Explaining variables/explaining constants
- Explaining comments
- Delete redundant comments



DESIGN IMPROVEMENT

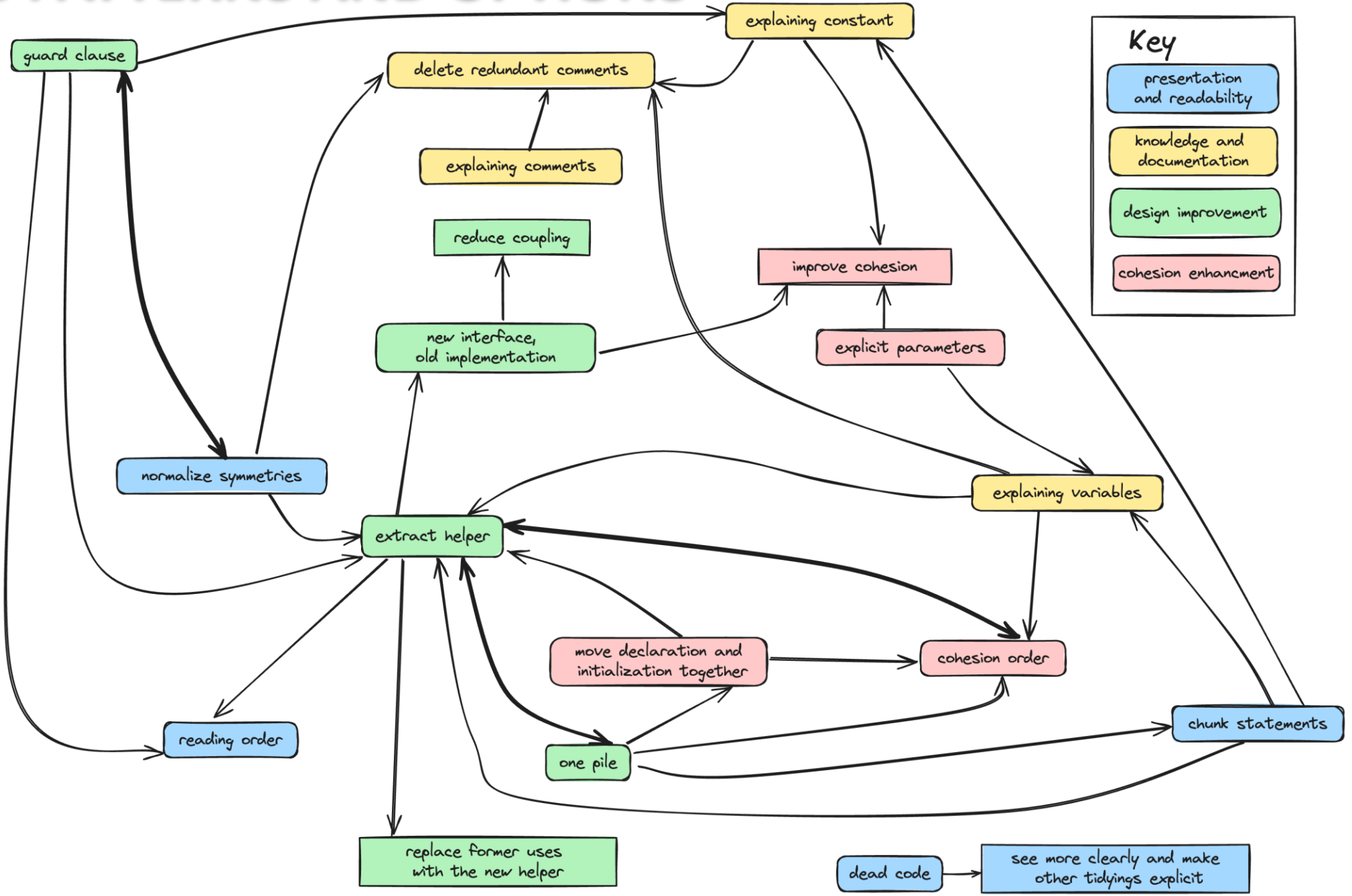
- Guard clause
- Extract Helper
- One pile (inline)
- New interface, old implementation

COHESION ENHANCEMENT

- Cohesion order
- Move declaration and initialization together
- Explicit parameters


LIVE CODING

TIDYING PATTERNS AND OPTIONS

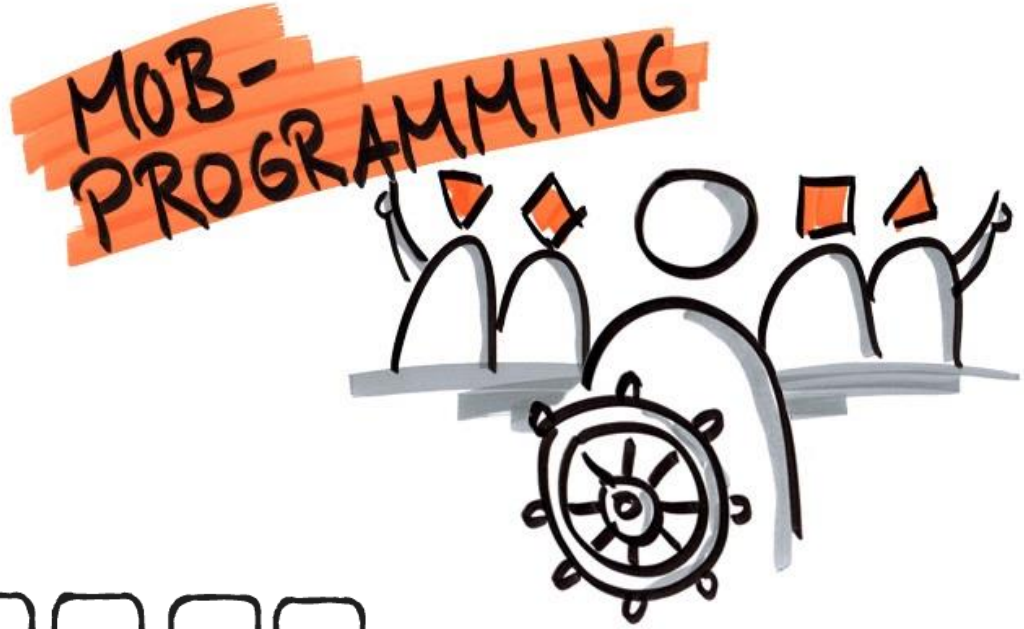


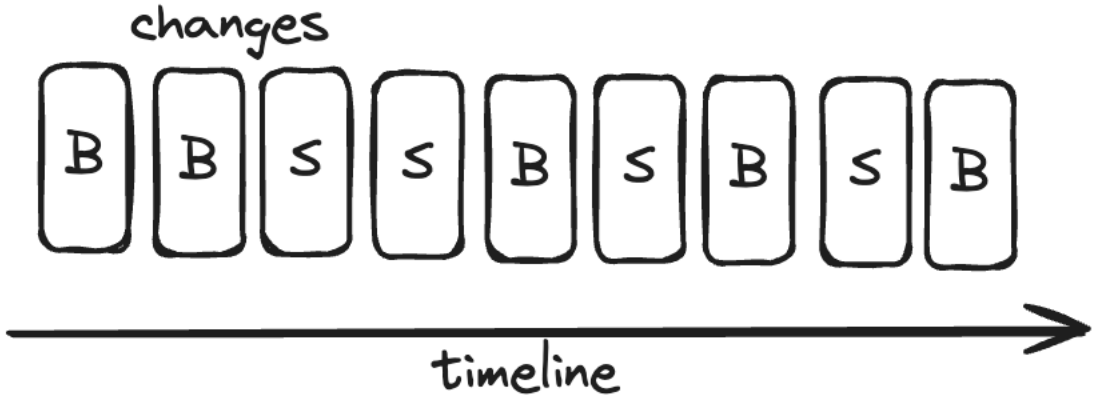
https://github.com/athiefaine/tidying_patterns/blob/main/tidying_patterns_options.excalidraw

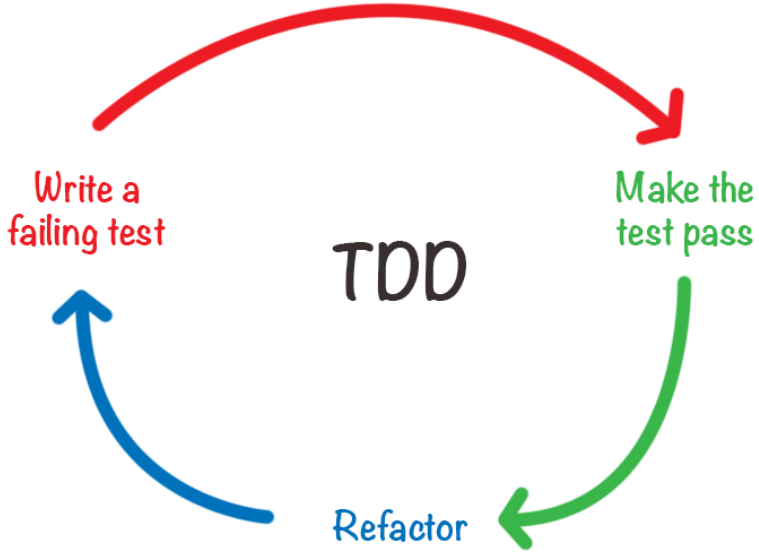

MANAGING
TIDYING
DAILY

STRUCTURE CHANGE VS BEHAVIOR CHANGE

https://softwareevolutivo.com.ec/mob-programming-el-arte-de-colaborar-con-el-equipo/



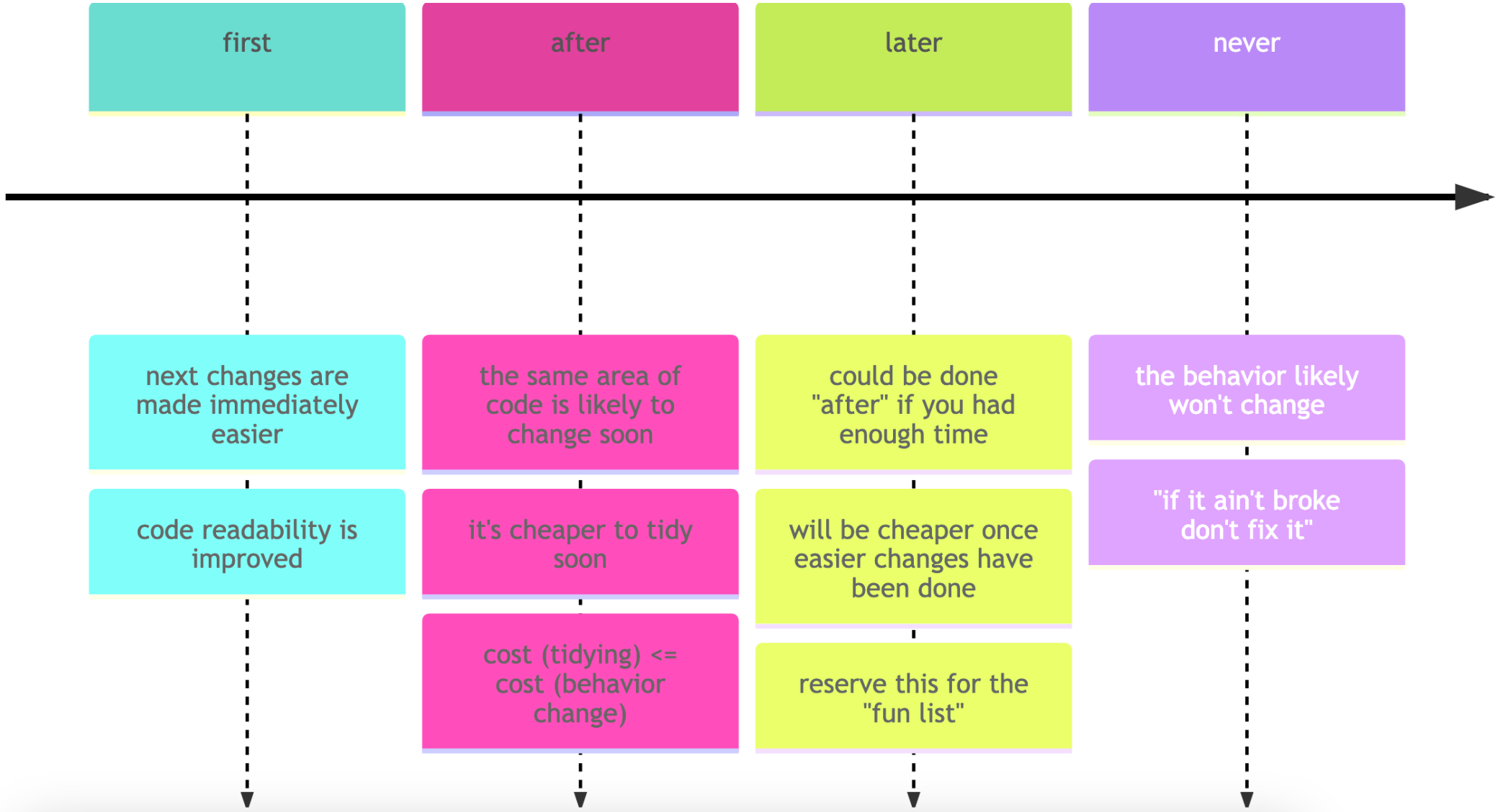





WHEN TO TIDY ?
A: FIRST
B: AFTER
C: LATER
D: NEVER

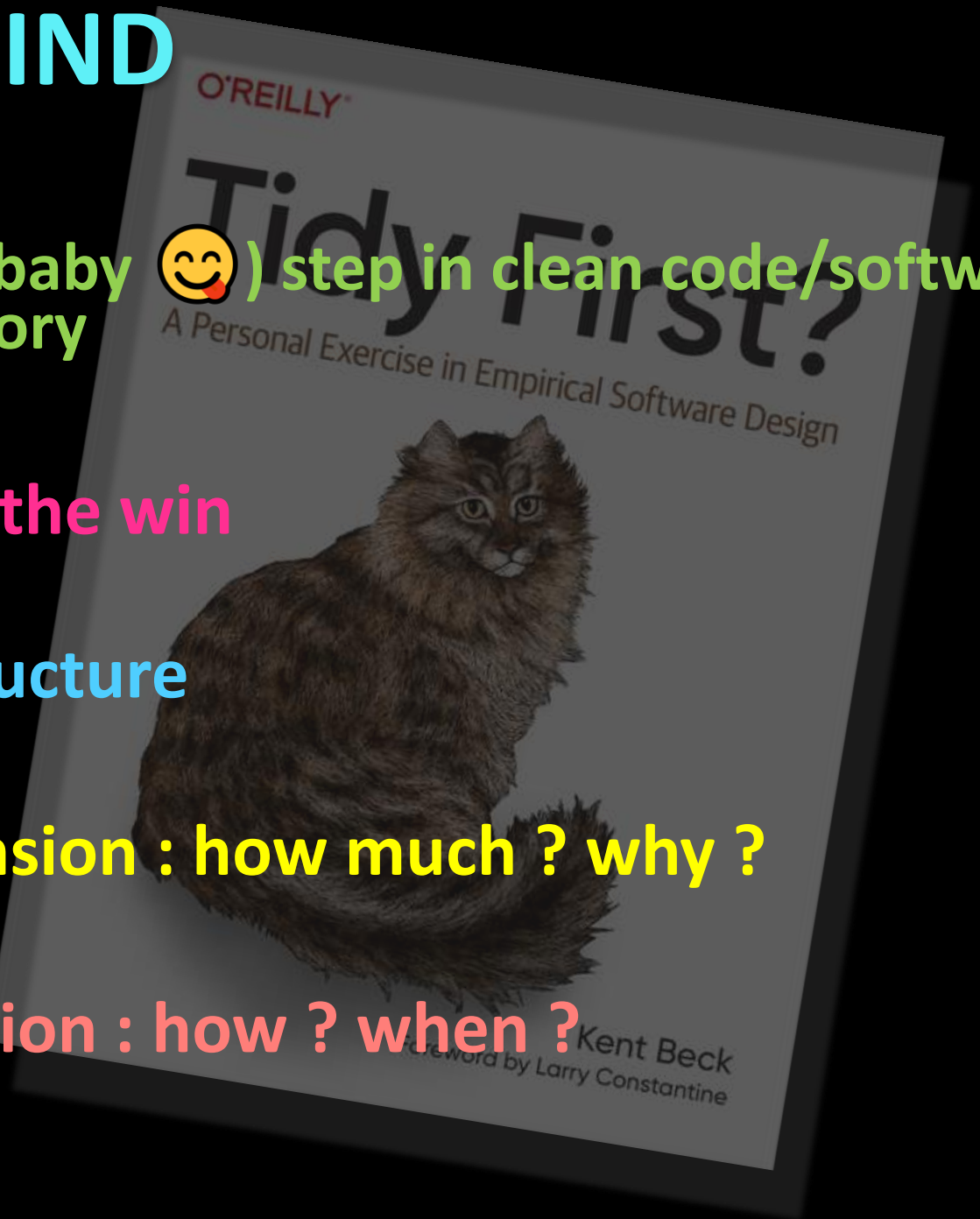

WHEN TO TIDY ?
A: FIRST
B: AFTER
C: LATER
D: NEVER

WHEN TO TIDY ?



https://github.com/athiefaine/tidying_patterns/blob/main/tidy_when.md

TO KEEP IN MIND



Another major (baby 😋) step in clean code/software engineering history

Optionality for the win

Behavior vs Structure

Strategic dimension : how much ? why ?

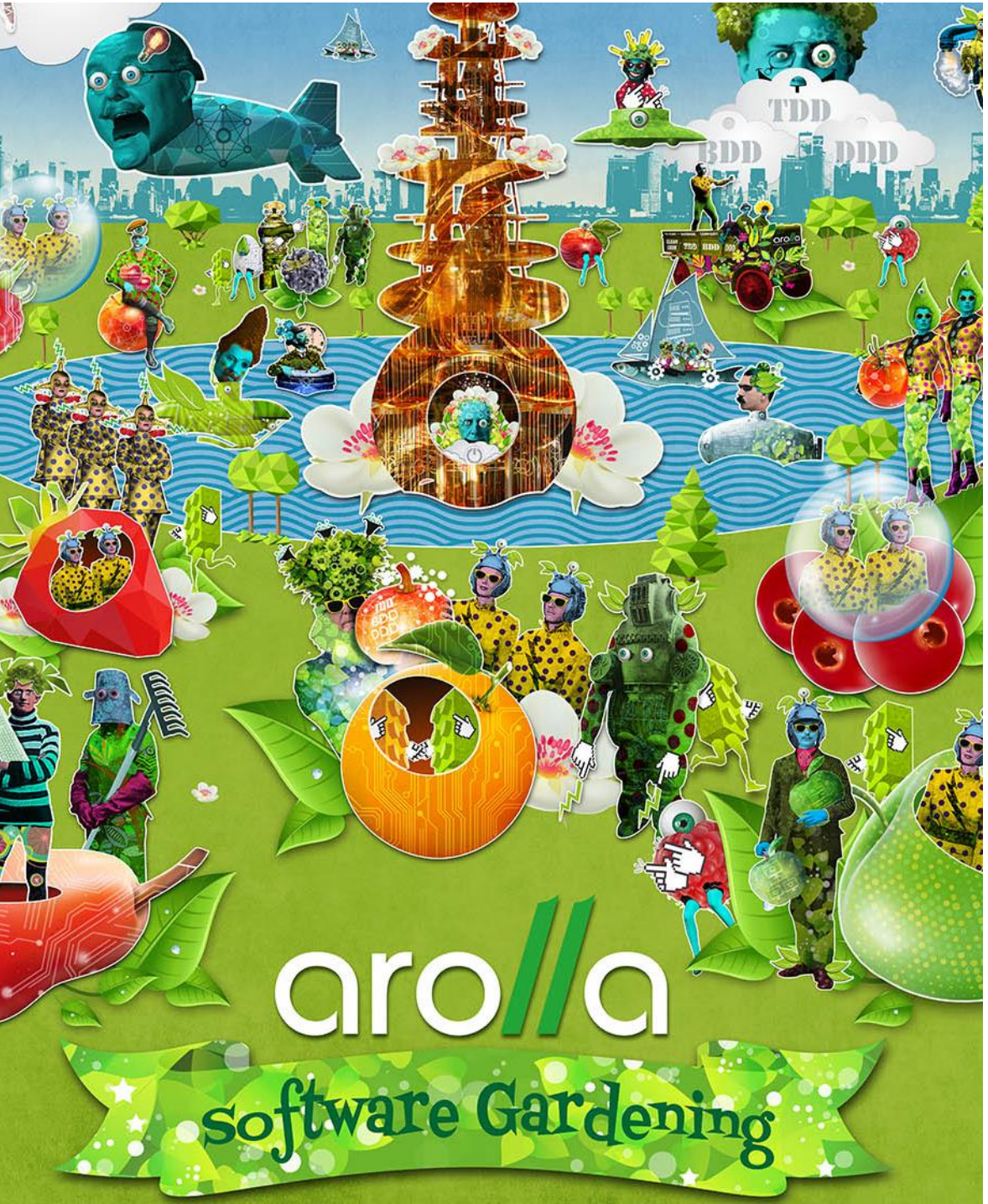
Tactical dimension : how ? when ?

TO BE CONTINUED …

individual developer

developer in a team

developer in an organization



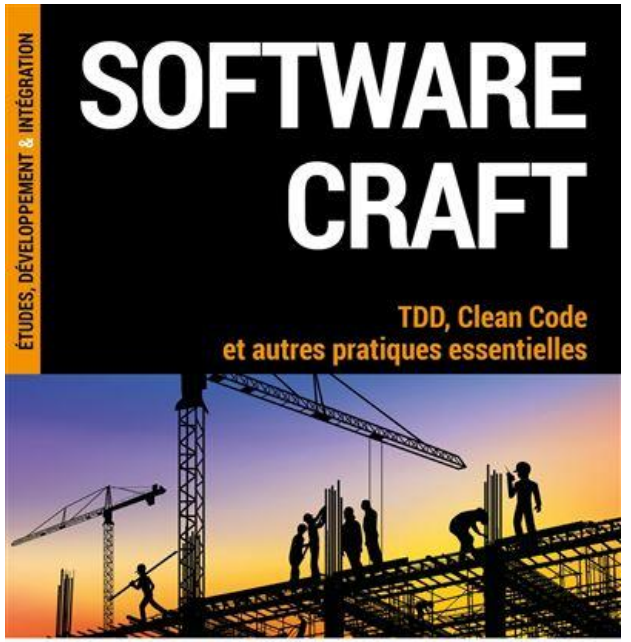
Thanks!

@arnaudthiefaine

athiefaine

arnaudthiefaine

Feedback ↓




Feedback ↓
QUESTIONS ?
A: FIRST
B: AFTER
C: LATER
D: NEVER