Martin Jonen Consulting

abap conf2025

mit ADventas Consulting

Principles to “beautify” your code

04. Juni 2025
Hamburg

Martin Jonen

16:30 - 17:30 CET
Channel 3
German

20% of IQ → Burnout → Unstructured code

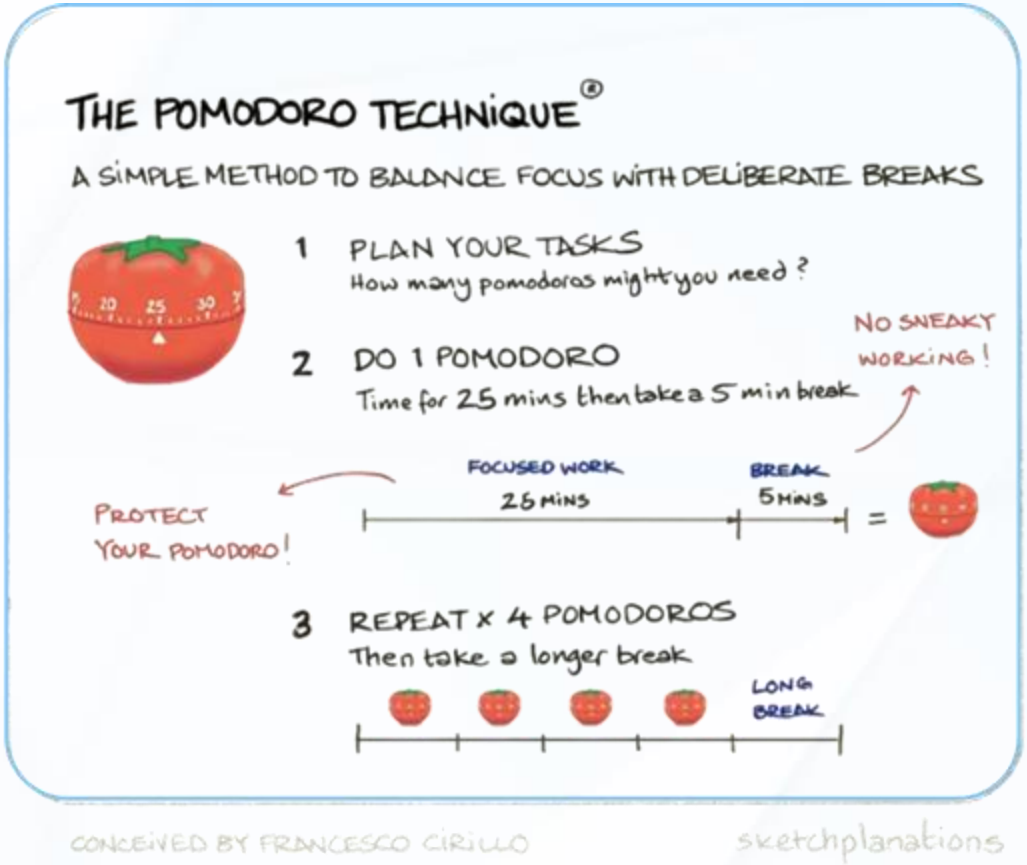
20-25 min. to “deep focus”

Principle #1 – Avoid distractions and gain focus

🎯 **Goal: High Focus => Higher code quality**

→ 20 – 25 min to “deep focus”

→ 20% IQ drop by “just” having phone on your table



Actions:

- Ban your “mobile phone” (flight mode, out of focus)
- Time fixed slots for focus work
  - Use **POMODORO**

Martin Jonen

SAP Consultant & Developer | SAP Software Engineering Coach

My journey so far …

- Freelance **SAP Consultant & Developer** (15+ years)
- Certified **'Exponential' SAP ABAP Software Engineering Coach**
  - **ASE Coaching** (www.ase-coaching.com)
    - Collaboration with Damir Majer & Prof. Dr. Christian Drumm
  - **ABAP Clean Coder Academy** (http://www.martinjonen.com/academy)
- **Author** (Online Courses)
- **Hobbies**: Running, Health, Yoga

At the end of this presentation

... You will have learned …

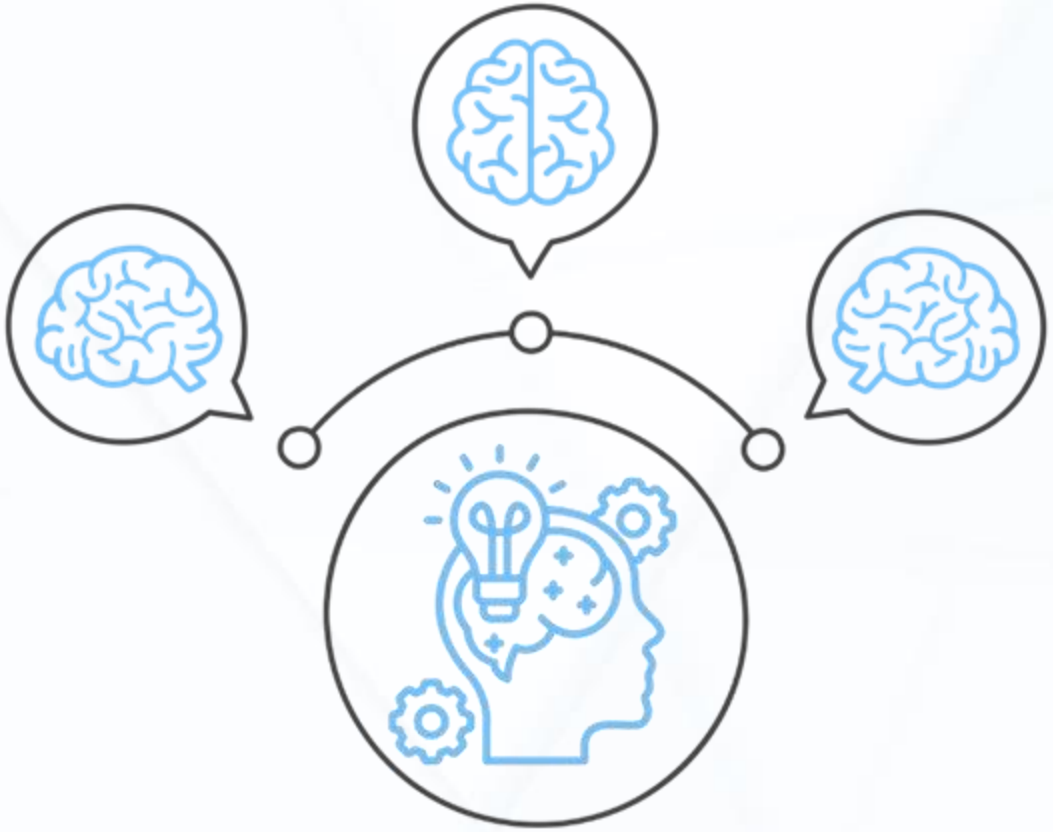
- **7 principles, which help …**

  - ☑ You structure your coding
  - ☑ You calm your mind and gain more focus
  - ☑ Your colleagues and future-self to read and understand your coding faster and easier (“audience in mind”)

Our TO-DO List:

- [x] #1 Avoid distractions and gain focus
- [ ] #2 Power of Three
- [ ] #3 Extract (functional) method
- [ ] #4 “Good” naming
- [ ] #5 Take a break
- [ ] #6 TEST first
- [ ] #7 Boy’s Scout Rule meets KISS

Principle #2 – Power of Three



Super Brain

Principle #2 – Power of Three

🎯 **Goal: Higher quality by challenging your ideas**

→ Higher code quality

→ Avoid cognitive overload

→ Learning field

Actions:

- Pair Programming
  → **POMODORO - ⏱ 25 min:** 1 screen, 7-min. keyboard/buddy/keyboard, set timer

- Design 3 different solutions up front

Together **E**veryone **A**chieves **M**ore

Our TO-DO List:

- [x] #1 Avoid distractions and gain focus
- [x] #2 Power of Three
- [ ] #3 Extract (functional) method
- [ ] #4 “Good” naming
- [ ] #5 Take a break
- [ ] #6 TEST first
- [ ] #7 Boy’s Scout Rule meets KISS

Principle #3 – Extract (functional) method

ZCL_VE_AVERAGE_RATING_MJ ▸ IF_SADL_EXIT_CALC_ELEMENT_READ~CALCULATE

```
METHOD if_sadl_exit_calc_element_read~calculate.
  DATA products TYPE STANDARD TABLE OF Z_C_Product_M_MJ.
  products = CORRESPONDING #( it_original_data ).

  " Read the ratings for the product(s) from the Z_C_Product_M_MJ entity
  READ ENTITIES OF Z_C_Product_M_MJ
    ENTITY Product BY \_Rating
    FIELDS ( Rating )
    WITH CORRESPONDING #( products )
    RESULT FINAL(ratings).

  " Calculate the average rating for each product by DB select
  SELECT Product,
         AVG( rating AS DEC( 2, 1 ) ) AS average_rating
    FROM @ratings AS r
    GROUP BY Product
    INTO TABLE @DATA(average_product_ratings).

  " Map average ratings to the products (output)
  LOOP AT products ASSIGNING FIELD-SYMBOL(<product>).
    READ TABLE average_product_ratings
      WITH KEY Product = <product>-ProductId
      INTO DATA(average_product_rating).
    IF sy-subrc = 0.
      <product>-AverageRating = average_product_rating-average_rating.
    ENDIF.
  ENDLOOP.

  ct_calculated_data = CORRESPONDING #( products ).
ENDMETHOD.
```

Principle #3 – Extract (functional) method

```
METHOD if_sadl_exit_calc_element_read~calculate.
  DATA products TYPE STANDARD TABLE OF Z_C_Product_M_MJ.
  DATA average_product_ratings TYPE tt_average_product_ratings.

  products = CORRESPONDING #( it_original_data )

  " Read the ratings for
  READ ENTITIES OF Z_C_Pr
    ENTITY Product BY \_R
    FIELDS ( Rating )
    WITH CORRESPONDING #(
    RESULT FINAL(ratings)

  " Calculate the average rating for each product by DB select
  SELECT Product,
         AVG( rating AS DEC( 2, 1 ) ) AS average_rating
    FROM @ratings AS r
    GROUP BY Product
    INTO TABLE @average_product_ratings.

  " Map average ratings to the products (output)
  products = map_average_ratings2products( average_product_ratings = average_product_ratings
                                           products                = products ).

  ct_calculated_data = CORRESPONDING #( products ).
ENDMETHOD.
```

```
METHOD map_average_ratings2products.

  result = products.

  LOOP AT result ASSIGNING FIELD-SYMBOL(<product>).
    <product>-AverageRating = average_product_ratings[ Product = <product>-ProductId ]-average_rating.
  ENDLOOP.

ENDMETHOD.
```

Principle #3 – Extract (functional) method

```
METHOD if_sadl_exit_calc_element_read~calculate.
  DATA(products) = CORRESPONDING tt_products( it_original_data ).

  DATA(average_product_ratings) = read_average_rating4products( products ).

  products = map_average_ratings2products( average_product_ratings = average_product_ratings
                                           products                = products ).

  ct_calculated_data = CORRESPONDING #( products ).
ENDMETHOD.
```

Fast entry points

Avoiding ‘cognitive overload’

Principle #3 – Extract functional method

Why “functional” method?

- Messaging (IN and OUT)
- Makes clear ***WHAT WE NEED*** to receive ***FEEDBACK***

Principle #3 – Extract functional method

🎯 **Goal: Create clear workflow**

→ Higher code quality

→ Avoid ‘cognitive overload’

- Information hiding

→ **S**ingle **R**esponsibility **P**rinciple (Do ***one*** thing and do it right)

Actions:

- If you can name the WHAT (***one*** task), extract it
- Use ADT in Eclipse quick fix “Extract method” (mark coding and press CTRL + 1)

Our TO-DO List:

- [x] #1 Avoid distractions and gain focus
- [x] #2 Power of Three
- [x] #3 Extract (functional) method
- [ ] #4 “Good” naming
- [ ] #5 Take a break
- [ ] #6 TEST first
- [ ] #7 Boy’s Scout Rule meets KISS

Principle #4 – “Good” Naming

Name the **WHAT**

```
" Map average ratings to the products (output)
products = map_average_ratings2products( average_product_ratings = average_product_ratings
                                         products                 = products ).
```

Comment the **WHY**

“Avg ratings should be displayed in the output

```
products = map_average_ratings2products( average_product_ratings = average_product_ratings
                                         products                 = products ).
```

Principle #4 – “Good” Naming

Use clear, specific names (make talking about coding effortless)

- LV_BUKRS → COMPANY_CODE
- LV_GPART → BUSINESS_PARTNER → CUSTOMER or VENDOR ?

Principle #4 – “Good” Naming

program_name

message

offset

source_code_list

```
REPORT ZSAGESSE_ABAP_CODE_INJ.
* Program Name
DATA: gv_prg_name TYPE sy-repid VALUE 'ZREPORT1'.
* Compiler Output
DATA: gv_msg TYPE string.
DATA: gv_line TYPE string.
DATA: gv_word TYPE string.
DATA: gv_off TYPE string.
* Source Code of dynamically created ABAP Program ( Code Injection Example )
DATA: gt_src TYPE STANDARD TABLE OF char1024.

* Add injected ABAP Source Code to internal table
APPEND |REPORT { gv_prg_name }.| TO gt_src.
APPEND |WRITE: / 'LogPoint for SAP Solutions protect your SAP Systems!'.| TO gt_src.

* Create a  (TYPE '1')- Executable ABAP Program,

INSERT REPORT gv_prg_name FROM gt_src PROGRAM TYPE '1' UNICODE ENABLING 'X'.

* Compile the dynamically created ABAP Program
GENERATE REPORT gv_prg_name MESSAGE gv_msg LINE gv_line WORD gv_word OFFSET gv_off.

IF sy-subrc = 0.
* Execute successfully generated ABAP program
  SUBMIT (gv_prg_name) AND RETURN.
ENDIF.

* Delete dynamically created ABAP Program
DELETE REPORT gv_prg_name.
```

Principle #4 – “Good” Naming

Be consistent

- All in German/English
- With/without ‘Hungarian Notation’
- Align with the team

```
METHOD if_sadl_exit_calc_element_read~calculate.
  DATA(products) = CORRESPONDING tt_products( it_original_data ).

  DATA(average_product_ratings) = read_average_rating4products( products ).

  products = map_average_ratings2products( average_product_ratings = average_product_ratings
                                           products                = products ).

  ct_calculated_data = CORRESPONDING #( products ).
ENDMETHOD.
```

In OO:

- **Members:** me->...
- **Locals:** company_code, business_partner_list (without ‘Hungarian Notation’)

Principle #4 – “Good” naming

🎯 **Goal: Reducing comments, make code “talkable”/ more understandable**

→ Name the **WHAT**, Comment the **WHY**

→ Be consistent

→ Align with the team

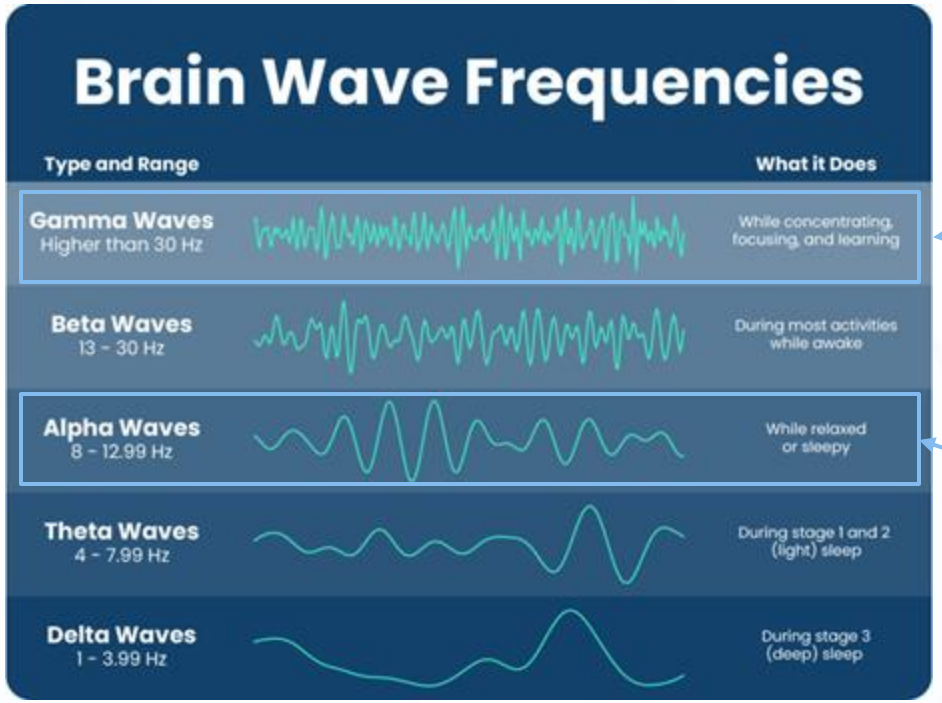
Actions:

- Apply the ‘WHAT and WHY technique’ to one method/class
- Talk about your coding. Is it “talkable”?
- Create a naming rulebook in your team

Our TO-DO List:

- [x] #1 Avoid distractions and gain focus
- [x] #2 Power of Three
- [x] #3 Extract (functional) method
- [x] #4 “Good” naming
- [ ] #5 Take a break
- [ ] #6 TEST first
- [ ] #7 Boy’s Scout Rule meets KISS

Principle #5 – Take a break



⏱ max. 40 min

Reduce stress

Most creative

20% increased brain activity when standing

1000% increase if doing sports

Principle #5 – Take a break

🎯 **Goal: Staying focused throughout the day**

→ Max. 40 min learning/working cycle

→ Alpha waves = creativity zone

→ Avoid being a “monitor zombie”

Actions:

- Take regular breaks (POMODORO)
- Move your body (“brain squats”, “cross hand to knee”)
- Take a walk (“fresh air”)

Our TO-DO List:

- [x] #1 Avoid distractions and gain focus
- [x] #2 Power of Three
- [x] #3 Extract (functional) method
- [x] #4 “Good” naming
- [x] #5 Take a break
- [ ] #6 TEST first
- [ ] #7 Boy’s Scout Rule meets KISS

Principle #6 – Test first

- Do not rush into programming
- What should your software achieve?

  → **Tests**

- **Sketch the solution (PEN & PAPER, best of Three)**

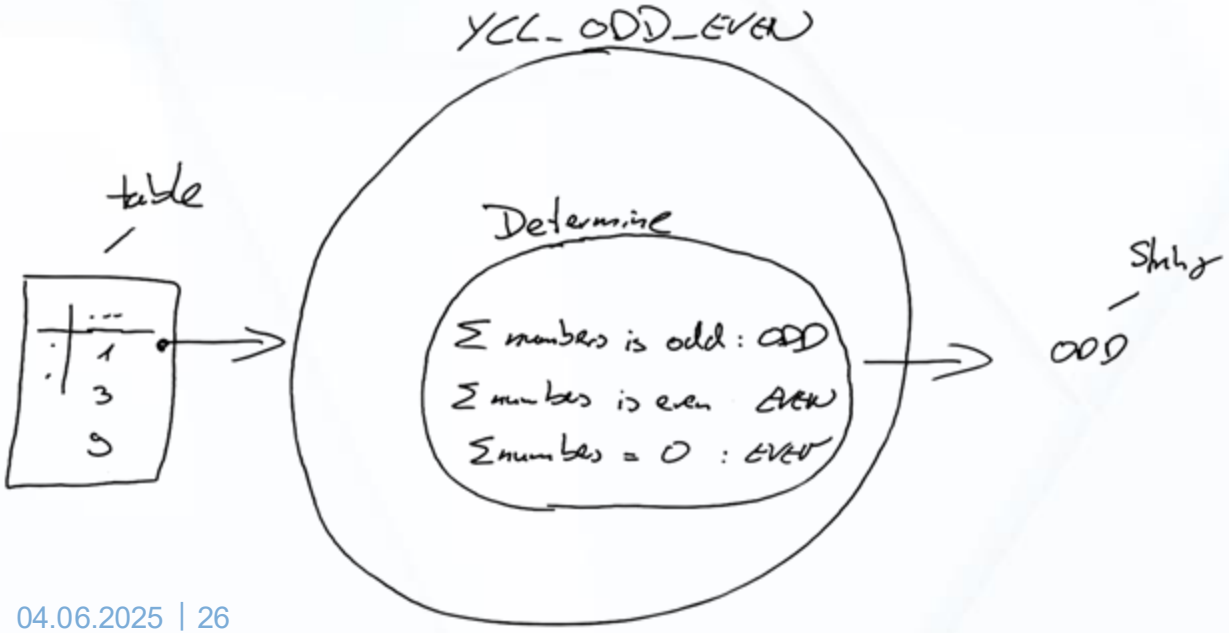
  → **Acceptance criteria (tests)**

Principle #6 – Test first

Even or Odd

Given an array of numbers, determine whether the sum of all the numbers is odd or even.

Give your answer in string format as ‘odd’ or ‘even’. If the input array is empty, consider it as: [0] (array with a zero).



Acceptance criteria:

- Sum is even → Output = “even”
- Sum is odd → Output = “odd”
- Sum is zero → Output = “even”

Principle #6 – Test first

🎯 **Goal: Focused, goal-oriented programming**

→ Focus on “solving the task”

→ Give your workflow structure and orientation

→ Less ‘deviation’ (YAGNI)

Principle #6 – Test first

Actions:

- Sketch the solution -> Acceptance criteria
- Optional: Create “scaffolding tests” → TDD

  e.g. Sum (1) is odd → Output = “odd”

Free TDD-Challenge:
https://www.martinjonen.com/tdd-3-day-challenge

Our TO-DO List:

- [x] #1 Avoid distractions and gain focus
- [x] #2 Power of Three
- [x] #3 Extract (functional) method
- [x] #4 “Good” naming
- [x] #5 Take a break
- [x] #6 TEST first
- [ ] #7 Boy’s Scout Rule meets KISS

Principle #7 – Boy’s Scout Rule meets K.I.S.S.

*"Always leave the **place** better than you found it."*

*"Always leave the **code** better than you found it."*
- Robert C. Martin

=> Invest 10% of time in refactoring

Principle #7 – Boy's Scout Rule meets K.I.S.S.

K.I.S.S.

→ **K**eep **I**t **S**imple, **S**tupid

→ "Do not overcomplicate things" → Find the simplest solution

Principle #7 – Boy's Scout Rule meets K.I.S.S.

🎯 **Goal: Clean code base by regularly investing time in refactoring**

→ "Leave it cleaner than you found it"

→ "Keep It Simple, Superstar"

Actions:

- Plan 10% time for refactoring
- Take the first baby step (smallest adjustment)

Our TO-DO List:

- #1 Avoid distractions and gain focus
- #2 Power of Three
- #3 Extract (functional) method
- #4 “Good” naming
- #5 Take a break
- #6 TEST first
- #7 Boy’s Scout Rule meets KISS

→

Stay organised

Calm your mind

Thank you for showing up!

✨ **Let's Make Clean ABAP Your New Standard**

If you're ready to:

- ✅ reduce bugs
- ✅ write testable, future-proof code
- ✅ grow without burning out…

🚀 **The ABAP Clean Coder Academy**

is open for **3 new members this month**

No pressure.
Just a conversation to see if it's the right fit for you.

Interested? Let's connect: 👇

📩 **martin@martinjonen.com**

🌐 ps://www.martinjonen.com/academy

in http://linkedin.com/in/martin-jonen/

Or simply talk to me after the session

Martin Jonen Consulting

abap conf2025

mit ADventas Consulting

Thank you!

Principles to “beautify” your code

Martin Jonen

16:30 - 17:30 CET

Channel 3

German