2026 CDISC China Interchange

From SAS to R: A Pioneer's Journey with Admiral

Implementing ADaM for a Prostate Cancer Study with PCWG-Modified RECIST

Ziwei Zhang

Speaker

Ziwei Zhang

- Senior Principal Statistical Programmer at Novartis
- 9+ years of clinical trial programming across multiple therapeutic areas, with a focus on immunology and oncology
- Extensive experience in SAS and R programming, including ADaM development and R/Admiral implementation

Agenda

01 Admiral & Project Context

02 From Learning to Application

03 Why Prostate Cancer Is Different

04 Real Application & Customization

05 Lessons Learned

What you'll take away

- How Admiral adapts to tumor-specific criteria
- Practical ideas for ADRS / ADTTE derivations
- How to customize Admiral without losing structure

Background & Context

Industry Context

- Pharmaverse & open-source R rising in clinical programming
- Growing industry experience with R-based regulatory submissions
- Admiral provides an industry-backed, CDISC-compliant ADaM solution

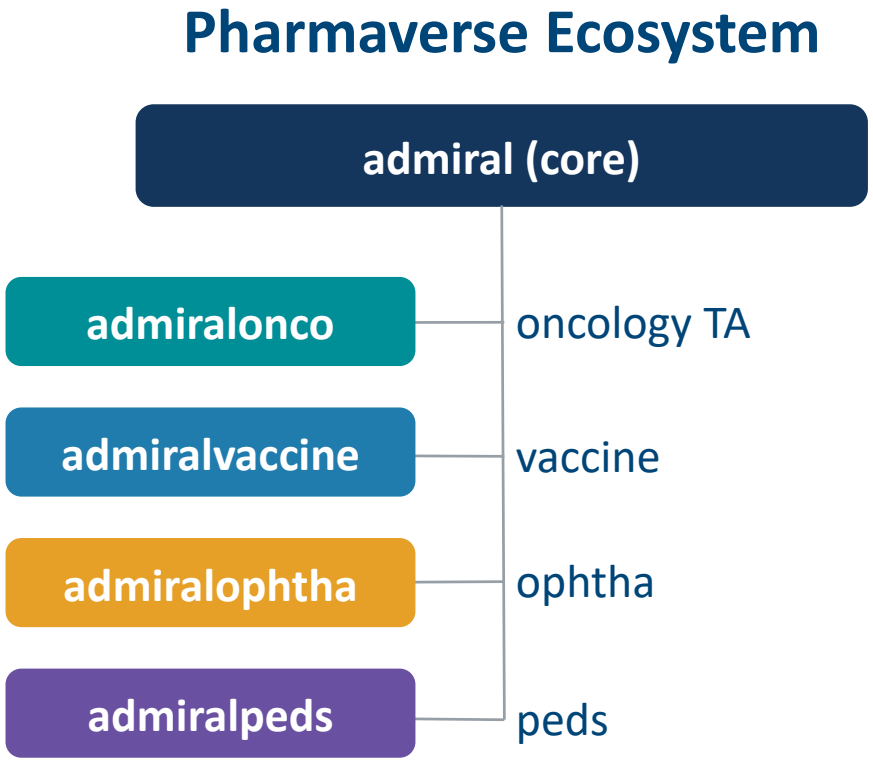
Our Context

- Novartis transitioning from SAS to R
- Selected as a pilot R-based programming project
- Prostate cancer study needed PCWG-modified RECIST
- Standard SAS macro did not fully apply

cdisc®

Admiral: ADaM in R Asset Library

- Open-source R package for ADaM dataset generation
- Developed collaboratively across pharma companies and CROs under the pharmaverse initiative
- CDISC-aligned, Modular, Traceable



My Learning Path

STEP 1: Explore

pharmaverse.github.io

- ✔ Read vignette
- ✔ Study examples
- ✔ Understand design

STEP 2: Replicate

Replicate on a pilot study

- ✔ Worked with known expected results
- ✔ Safe environment to experiment
- ✔ Applied admiral to a standard RECIST scenario

STEP 3: Apply

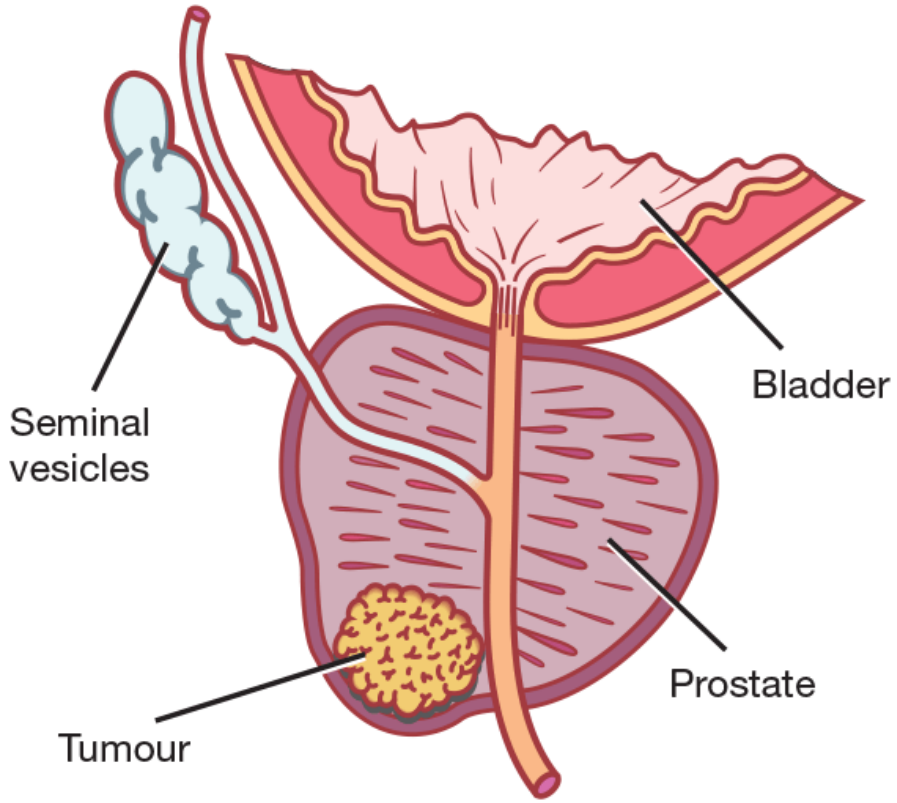
Real project: Prostate cancer study

- ✔ Adapt to PCWG
- ✔ Iterate & refine

cdisc®

Prostate Cancer: Disease Characteristics

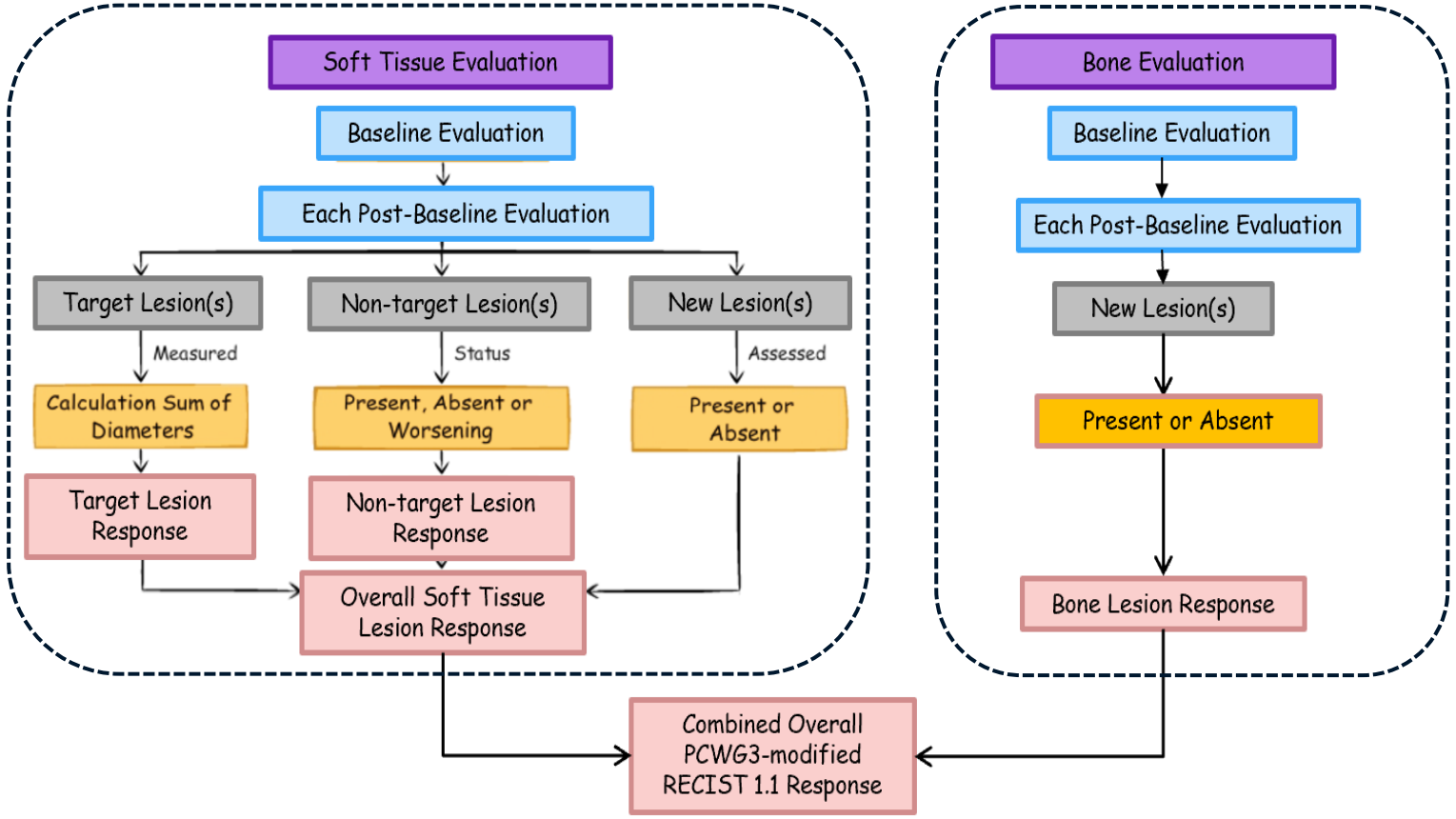
- **Epidemiology*:**
  - The most-commonly diagnosed cancer
  - The fourth most common cause of death in worldwide
- **Disease Characteristics:**
  - Prostate cancer commonly metastasizes to both soft tissue and bone
  - Bone lesions are a major component of disease burden and disease progression
  - Assessment of bone disease differs from assessment of measurable soft-tissue lesions



* Reference: https://www.who.int/news-room/fact-sheets/detail/cancer

Why Response Assessment Is Different in Prostate Cancer

- **Impact on Response Evaluation:**
  - Standard RECIST primarily focuses on measurable soft-tissue lesions
  - Bone disease requires a separate assessment approach
  - Soft-tissue and bone assessments must be combined to determine overall response



Our Approach: From Decision to Implementation

The Situation

- Team moving toward R
- PCWG-modified RECIST needed
- Existing SAS RECIST macro → optimized for standard RECIST, not designed for PCWG

Options We Considered (team discussion)

Option 1: Adapt existing SAS macro

- Macro optimized for standard RECIST
- Extending to PCWG would require significant rework

Option 2: Build from scratch in R

- Maximum flexibility, but reinvents the wheel

Option 3: Adopt Admiral + customize ✓

- CDISC-compliant foundation
- Modular & flexible for PCWG logic
- R-native, aligned with team's direction

A Deeper Realization

- Starting with R + Admiral gave us the flexibility
  - design ADaM around PCWG logic
  - rather than force PCWG into a macro built for a different scenario

Real Application: The Three Datasets I Worked On

ADRS

Response Analysis

- ✔ Best Overall Response (BOR)
- ✔ PCWG-modified RECIST logic
- ✔ admiral event() framework

ADTR

Tumor Results

- ✔ Diameter-based
- ✔ Custom code (not covered today)

ADTTE

Time-to-Event

- ✔ PFS, OS, and other endpoints
- ✔ admiral + custom wrappers

cdisc®

ADRS: PCWG-Modified RECIST

- **Why is response derivation harder for prostate cancer?**

💡 Two response tracks must be evaluated separately, then combined

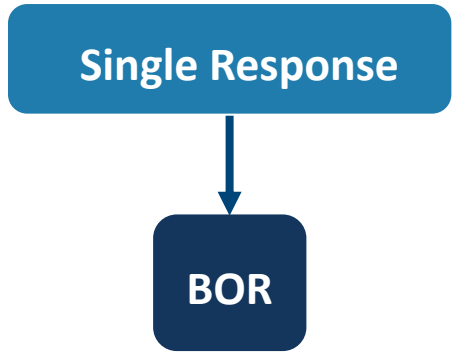
- a logic beyond standard RECIST

👉The modular framework in admiral fits this pattern well

- Combine response assessments, define each response outcome as an event, then apply the project-defined priority to derive BOR

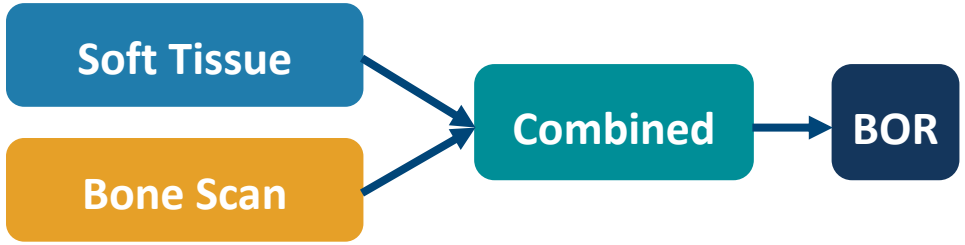
Standard RECIST

A single response track based primarily on measurable (mostly soft tissue) lesions



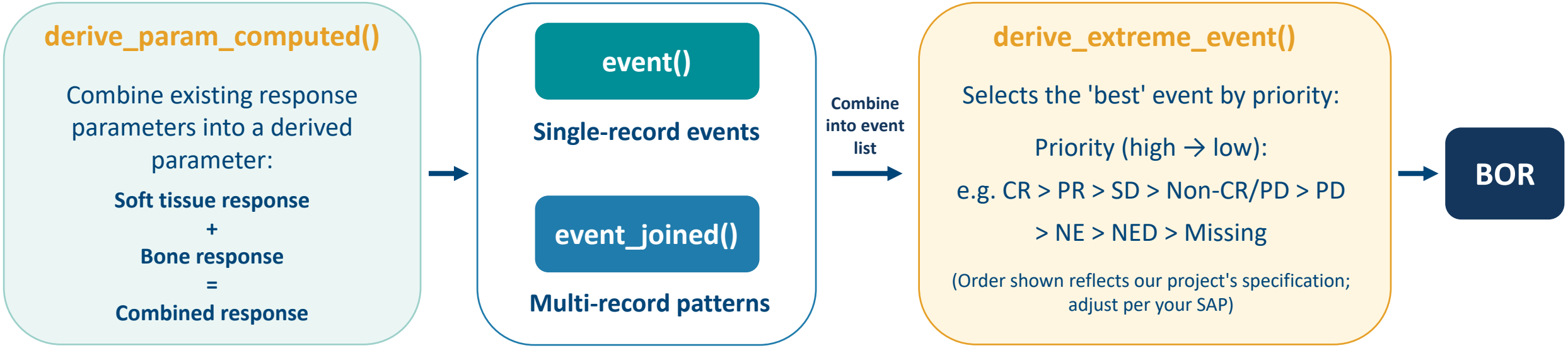
PCWG-Modified RECIST

Two parallel tracks: soft tissue response AND bone scan response, must be evaluated and combined into an overall response.



ADRS: A Modular Derivation Framework

- **The Design Pattern: Define components → Define events → Combine into BOR**



Where the project-specific logic lives:

- Response combination → derive_param_computed()
- Event definitions → event() & event_joined()
- BOR prioritization → derive_extreme_event()

ADRS: Customizing the Framework

Example 1: Confirmation with a project-specific tolerance rule

- ✔ Our project allows one intervening SD between two PRs, per SAP
- ✔ Sequence: PR → SD → PR → PR
- 👉 event_joined() + count_vals() can match this pattern → ☑ Confirmed PR

Example 2: Special responses (event)

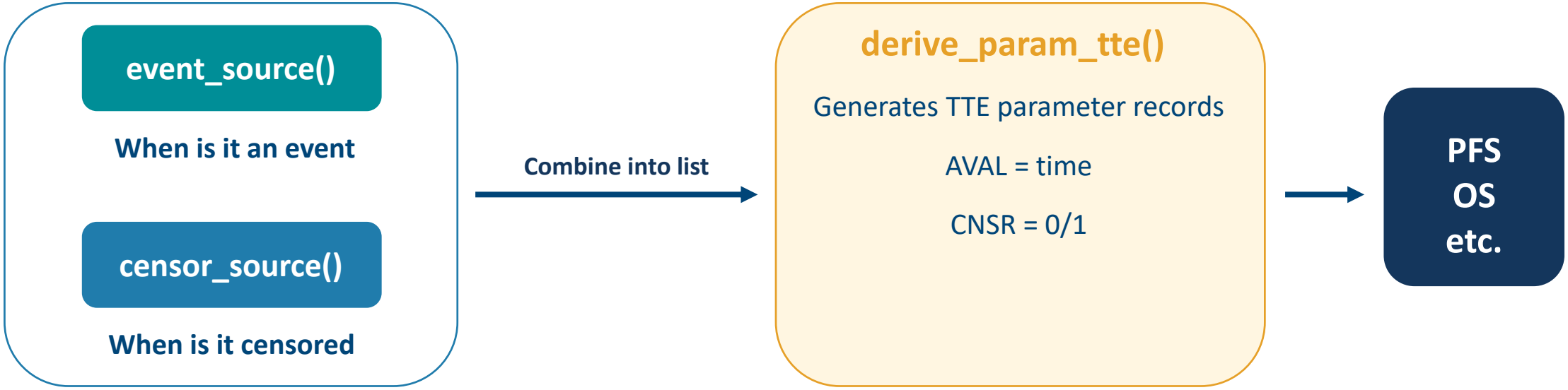
- ✔ PDu - Unconfirmed Progressive Disease
- ✔ NED - No Evidence of Disease
- 👉 Each defined as its own event() with tailored conditions
  → All events plug into the same derive_extreme_event() pipeline

cdisc®

💡 The framework accommodates project-specific response patterns without changing the overall derivation pipeline

ADTTE: Reusing the Modular Pattern

- **Similar design philosophy as ADRS: Define sources → Derive parameters**



- ✔ Standard, well-documented pattern
- ✔ Flexible for defining any TTE endpoint - but the source setup is repetitive across endpoints

🤔 With many endpoints (PFS, OS, TTP, DOR, ...), writing the source lists gets... tedious

Going Further: Building Wrappers on Top of admiral

⚠ **Opportunity for a Convenience Layer:**

- derive_param_tte() requires:
  - Each event/censor defined separately
  - Manually organized into TWO separate lists:
    (events = list(...), censors = list(...))
- With many events/censors, maintaining two parallel source lists created an avoidable organizational risk for our team

Wrapper 1

Build source (event/censor)
- One unified function
- Type-tagged: event or censor
- All definitions live in ONE combined list

Wrapper 2

- Auto-splits by Type
- Routes to correct arguments of derive_param_tte()

cdisc®

Going Further: Building Wrappers on Top of admiral

Why we built

- ✔ Multiple TTE endpoints meant repeatedly organizing large numbers of event and censor definitions

What improved

- ✔ Fewer chances to misplace event/censor definitions
- ✔ Cleaner code organization (one list, not two)

What did not change

- ✔ Individual definitions are still verbose
- ✔ They are convenience layers, not new capabilities

Why it matters

- ✔ Modular design allows targeted workflow improvements without changing the underlying derivation approach

cdisc®

Key Lessons Learned

Structure is a strength

For standard, well-defined endpoints (like oncology response), admiral's structure promotes consistency, traceability, and CDISC alignment

Complex logic is manageable

Even for complex cases like PCWG, Admiral gave us enough flexibility to adapt, without rebuilding everything from scratch

✔ The sweet spot: constrained yet adaptable

Modularity comes with trade-offs

Admiral works like building blocks: define each piece clearly, then assemble.

✔ Each step is transparent and extensible

⚠ Code volume increases: especially for endpoint-heavy datasets like ADTTE

cdisc®

Acknowledgement

Special thanks to the many colleagues who supported this journey from learning to real-world application:

- Study team members for the opportunity to apply Admiral in a complex prostate cancer study
- Programming and statistics colleagues for their partnership and feedback
- Everyone who contributed ideas, reviews, and encouragement along the way

Thank You!