The AC/DC Framework: A Recipe Analogy

The Big Picture

Think of clinical trial analysis like cooking. Every time you want to prepare a dish, you need three things:

1. **A Recipe** (Template) - The pattern that describes what you're making
2. **Ingredients** (Semantic Transformation Elements) - The components that go into the dish
3. **Cooking Instructions** (Method) - How to combine the ingredients

Just as a recipe for vinaigrette doesn't care whether you use organic extra-virgin olive oil from Tuscany or store-brand vegetable oil—it just says "oil"—the AC/DC Framework separates the *concept* of what you need from the *specific product* you actually use.

The Components

 The Recipe (Template)

A **Template** is like a recipe card. It defines:

- What dish you're making (clinical intent)
- What ingredients you need (which Semantic TEs)
- What role each ingredient plays (cube roles)
- Which cooking method to use (references a Method)

Example Recipe: "Classic Vinaigrette"

```
Ingredients needed:
  - oil (3 parts)
  - vinegar (1 part)
  - seasoning (to taste)

Method: Emulsification

Serves: Salad dressing
```

Example Template: "ChangeFromBaseline"

Semantic TEs needed:
- analysis_value (input measure)
- baseline_value (input measure)
- change_value (output measure)

Method: Subtraction

Produces: Derived change variable

The recipe doesn't know or care about brands—it works with abstract ingredient types. Similarly, the Template doesn't know about CHG, AVAL, or BASE—it works with semantic concepts.

The Ingredients (Semantic Transformation Elements)

Semantic TEs are like ingredient categories in a recipe. The recipe calls for "oil"—not "Bertolli Extra Virgin Olive Oil, 500ml bottle, purchased from Whole Foods."

Recipe Ingredient	Semantic TE	What It Represents
Oil	analysis_value	The value being analyzed
Vinegar	baseline_value	The reference point
Seasoning	treatment_planned	The grouping factor
The final dressing	change_value	The derived result

Just as "oil" could be:

- Olive oil from Italy
- Canola oil from Canada
- Coconut oil from Thailand

The Semantic TE `analysis_value` could be:

- `AVAL` in ADaM
- `STRESN` in SDTM
- `value_as_number` in OMOP

The recipe works the same way regardless of which specific oil you use.

👨‍🍳 The Cooking Instructions (Method)

A **Method** defines HOW to combine ingredients—the technique, not the specific items. It uses named roles (placeholders) that get filled by actual ingredients.

Cooking Method: "Emulsification"

```
Instructions:
  1. Place [acid] in bowl
  2. Slowly drizzle [fat] while whisking
  3. Add [seasonings] and mix
  4. Result: [emulsion]

Roles:
  - acid (input) → receives: vinegar
  - fat (input) → receives: oil
  - seasonings (input) → receives: salt, pepper, mustard
  - emulsion (output) → produces: vinaigrette
```

Derivation Method: "Subtraction"

```
Formula:
  difference := minuend - subtrahend

Roles:
  - minuend (input) → receives: analysis_value
  - subtrahend (input) → receives: baseline_value
  - difference (output) → produces: change_value
```

The method doesn't care what specific oil or vinegar you use—it just knows how to combine "fat" and "acid" into an emulsion. Similarly, the Subtraction method doesn't care about variable names—it just knows how to subtract the "subtrahend" role from the "minuend" role.

Putting It Together: From Recipe to Plate

Layer 1: The Recipe Card (Template + Instance)

You decide to make vinaigrette for tonight's salad. You pull out the recipe card (Template) and note what you need.

Tonight's Vinaigrette (Instance):
Using recipe: Classic Vinaigrette
For: Caesar salad
Serving: 4 people

Layer 2: The Pantry (Clinical Transformation Model)

Your pantry has a labeling system. The jar labeled "OIL" could contain different products depending on when you bought it:

Pantry Labels (CTM):
"OIL" → Currently contains: Olive oil
"VINEGAR" → Currently contains: Balsamic vinegar
"SEASONING" → Currently contains: Dijon mustard

In AC/DC terms:

CTM Bindings:
ADaM: AVAL → analysis_value
ADaM: BASE → baseline_value
ADaM: CHG → change_value

Layer 3: The Actual Products (Implementation)

When you open the pantry, you grab the actual physical items:

What's Actually in the Jar:
"OIL" jar → Bertolli Extra Virgin, Lot #2024-03
"VINEGAR" jar → Whole Foods 365 Balsamic

In AC/DC terms:

Study Implementation:
AVAL → ADQSADAS.AVAL (physical column in actual dataset)
BASE → ADQSADAS.BASE
CHG → ADQSADAS.CHG

The Magic: Same Recipe, Different Kitchens

Scenario 1: Your Home Kitchen (ADaM)

Recipe: Vinaigrette
Pantry system: Home labels
Actual products: What's in your fridge

→ Result: Homemade dressing with your ingredients

Template: ChangeFromBaseline
Standard: ADaM
Study: ADQSADAS

→ Generated: CHG = AVAL - BASE

Scenario 2: Restaurant Kitchen (OMOP)

Recipe: Vinaigrette (same recipe!)
Pantry system: Commercial kitchen labels
Actual products: Restaurant suppliers

→ Result: Same dressing, different source ingredients

Template: ChangeFromBaseline (same template!)
Standard: OMOP
Study: measurement table

→ Generated: change = value_as_number - baseline_value

The recipe didn't change. Only the pantry labeling and actual products changed.

Why This Matters

Approach 1: No Recipe at All (Jigsaw Puzzle)

Every time you want vinaigrette, you write out the full instructions from scratch:

"Take the Bertolli Extra Virgin Olive Oil 500ml from the second shelf and pour 3 tablespoons into the blue ceramic bowl, then take the Whole Foods 365 Balsamic from the door shelf..."

This is like a **jigsaw puzzle** - every piece has exactly one place, and together they make exactly one picture. Skilled work, satisfying to complete, but you'll never rearrange those pieces into a different dish.

Problems:

- Move to a new kitchen? Rewrite everything.
- Different brand of oil? Find and replace throughout.
- Share with a friend? They need your exact setup.
- Every dish is assembled anew, every time.

Approach 2: Proprietary Recipe Systems (Incompatible LEGO)

Some organisations recognised the jigsaw problem and started building their own recipe systems:

Company A's Recipe System (YAML + R):

```yaml
dish:
  name: Vinaigrette
  ingredients:
    - oil: '!is.na(olive_oil)'  # R syntax
  steps:
    mix_step:
      code_id: "emulsify_v2.R"  # external code file
```

Company B's Recipe System (Custom DSL):

```
if PANTRY.OIL_TYPE = 'OLIVE'
    and the freshness of
        ([PANTRY.PURCHASE_DATE + 30 days, PANTRY.EXPIRY])
    is not later than TODAY,
then use PANTRY.OIL,
else use PANTRY.BACKUP_OIL.
```

This is progress - they're trying to build with **LEGO bricks** instead of jigsaw pieces. But here's the challenge: **every company has built their own LEGO system with proprietary brick sizes.**

Problem	What It Means
Proprietary	Company A's recipes don't work in Company B's kitchen
Platform-specific	Company A's system only works with R; Company B needs a custom interpreter
Spec ≠ Code	The "recipe" is just a pointer to actual code hidden in external files
Not portable	Can't share recipes across companies, CROs, or regulatory agencies
Missing intent	Tells you HOW to mix, but not WHY you're making this dish

Your recipes won't work in my kitchen. We've created a hundred incompatible cookbooks, each written for a different set of equipment and measurement systems.

Approach 3: A Common Recipe Framework (AC/DC)

What we need isn't just any LEGO system - we need **standard brick sizes** that everyone agrees on. Not static standards that prescribe every detail, but templates that define the *shape* of what you need and let your study context fill in the specifics.

```
Recipe: Vinaigrette
  Ingredients (by role):
    - fat: oil
    - acid: vinegar
    - flavouring: seasoning
  Method: Emulsification

  # The template defines shapes, not colours
  # Your oil might be olive, mine might be canola
  # Same recipe works for both
```

Think of it as **LEGO instructions that say "place a 2x4 brick here."** In your study, that brick might be red (AVAL). In mine, it's blue (STRESN). The template is shared; the configuration is yours.

Benefits:

- **Any kitchen works** (standard-independent)
- **Any brand works** (implementation-independent)
- **Share freely** (portable across organisations)
- **Consistent results** (reproducible)
- **Intent is clear** (we know WHY we're making vinaigrette, not just HOW)

"But Will the Vinaigrette Taste the Same?"

This is a fair question. If I use Tuscan olive oil and aged balsamic, and you use canola oil and white wine vinegar - we'll get different vinaigrettes. **And that's okay.**

What the Recipe Guarantees (And What It Doesn't)

Let's be clear about what we are NOT doing: We are not telling chefs what to cook.

Choosing the right dish for the occasion is a skilled decision. A dinner party might call for a delicate beurre blanc; a summer barbecue needs a robust chimichurri. That choice belongs to the chef - it requires expertise, judgement, and knowledge of the context.

What we *are* doing is providing a **well-organised cookbook** so that once the chef decides "I'm making vinaigrette," they have a clear, reliable, shareable template to work from.

Similarly, **we are not standardising how statisticians choose their analytical model.** Choosing the right model for a study is critical scientific work:

- Should we use MMRM or ANCOVA?
- Is a Cox model appropriate, or do we need a competing risks approach?
- How many covariates are needed to control for confounding?
- Should we include treatment-by-time interactions?

These decisions require statistical expertise, understanding of the disease area, and knowledge of regulatory expectations. **That judgement remains entirely with the statistician.**

What the AC/DC Framework provides is a **library of well-defined templates** so that once the statistician decides "I'm using ANCOVA with baseline and site as covariates, no interactions," they have a clear, reliable, shareable way to specify and execute that choice.

> **We're not standardising the chef's judgement. We're standardising the cookbook they choose from.**

The recipe guarantees **structural consistency**, not identical output - and it allows for **chef's choices** within the framework:

What's Standardised	What's Configurable	What Varies by Study
It's a vinaigrette (emulsion of fat + acid)	How many herbs to add	Which brand of oil
The base technique is emulsification	Whether to add mustard for extra emulsification	Your pantry contents
The ratio of fat to acid	Whether to include garlic	The final flavour
How to combine the ingredients	Seasoning intensity	Whether your guests like it

Think of it like this: the recipe says "Vinaigrette" and specifies the emulsification technique. But it doesn't dictate:

- **Number of seasonings:** "Add 2-4 seasonings of your choice" (like choosing how many covariates)
- **Optional ingredients:** "For a creamier result, add mustard" (like choosing whether to include interactions)
- **Intensity levels:** "Season to taste" (like setting alpha levels or confidence intervals)

We're not standardising your culinary choices - we're standardising how those choices are expressed and executed.

A chef in Paris might make:

```
Vinaigrette (Template: Classic_Vinaigrette)
  - Base: olive oil + red wine vinegar
  - Seasonings: salt, pepper, herbs de Provence  [3 covariates]
  - Extras: Dijon mustard                        [interaction: YES]
```

A chef in Tokyo might make:

```
Vinaigrette (Template: Classic_Vinaigrette)
  - Base: sesame oil + rice vinegar
  - Seasonings: salt, ginger                     [2 covariates]
  - Extras: none                                 [interaction: NO]
```

Both are valid instances of the Vinaigrette template. The framework doesn't judge whether three seasonings is better than two, or whether mustard should be included. It simply ensures that:

1. Both chefs are making the same *type* of thing (vinaigrette)
2. Both are using the same *technique* (emulsification)
3. Both can clearly *communicate* their choices (structured configuration)
4. Anyone can *reproduce* either dish (traceable specification)

Similarly, the AC/DC Framework guarantees:

What's Standardised	What's Not
An ANCOVA will be performed	Your specific p-value
Change from baseline will be calculated	The magnitude of change in your study
The model includes baseline, site, treatment	Your study's actual coefficients
Results are traceable to inputs	Whether your drug works

The Output *Should* Be Different

Think about it: if two studies using the same ANCOVA template got identical results, something would be very wrong! The whole point is that:

- **The recipe is the same** → Reproducible methodology
- **The ingredients differ** → Your study's actual data
- **The output differs** → Your study's actual results

What we're standardising is the **process**, not the **findings**.

What "Same" Really Means

When a regulator reviews two submissions that both used the CFB_ANCOVA template, they can trust:

1. **Same analytical approach** - Both performed ANCOVA with baseline covariate adjustment
2. **Same structural assumptions** - Both used Type III sums of squares, same degrees of freedom method
3. **Same traceability** - Both can show exactly how results link back to source data
4. **Comparable interpretation** - The LS Mean difference means the same thing in both

They're not expecting identical numbers - they're expecting **methodological consistency** and **transparent provenance**.

A Michelin Star Analogy

A Michelin-starred restaurant in Paris and one in Tokyo might both serve "French onion soup." The recipes follow the same classical technique. But:

- Paris uses sweet Vidalia onions; Tokyo uses local Japanese onions
- Paris uses Gruyère; Tokyo might use a local alpine-style cheese
- The broths reflect local beef and water

Both are *authentically* French onion soup. Both follow the *same method*. Neither is "wrong" - they're **faithful implementations** of a shared recipe adapted to local ingredients.

That's exactly what AC/DC enables: **faithful implementations** of shared analytical methods, adapted to each study's data.

> **The recipe doesn't promise your soup will taste like mine. It promises we both made soup the same way, and we can both explain exactly how.**

Summary Table

Cooking Concept	AC/DC Component	What It Does
Recipe	Template	Defines the pattern, references ingredients by role
Ingredient type (e.g., "oil")	Semantic TE	Abstract concept, not tied to specific product
Cooking method (e.g., "emulsify")	Method	Instructions using named roles
Pantry label	CTM Binding	Maps concept to standard (ADaM, OMOP)
Actual product	Implementation	Physical variable in actual dataset
Tonight's dish	Instance	Specific application with fixed context
"For Caesar salad, 4 servings"	Slice	The specific analysis context

---

The Key Insight

> **A good recipe works in any kitchen, with any brand of ingredients, as long as you have the right *types* of ingredients and follow the method.**

Similarly:

> **A good AC/DC Template works with any data standard, any study, as long as you have the right *types* of data concepts and apply the appropriate statistical method.**

The recipe doesn't need to know if your olive oil is organic, cold-pressed, or from a specific region. It just needs "oil" that plays the role of "fat" in the emulsification process.

The Template doesn't need to know if your data is in ADaM, SDTM, or OMOP. It just needs a concept that plays the role of "analysis_value" in the calculation.

That's the power of separation of concerns.