PHUSE SDE BASEL · 2026

Bringing Mock Shells to Life

Supervised AI automation of the clinical TLG workflow with DPS2R — keeping the human in the loop (HITL) at every iteration

DPS2R

Guillaume Abgrall · Principal II Statistical Programming Lead

Tadeusz Lewandowski · Head of Data Engineering

J&J Innovative Medicine

THE PROBLEM

Mock shells describe intent — not implementation

DPS (Data Presentation Specifications) = mock shells augmented with programming notes. A gap still remains.

DPS mock shell

Defines **WHAT** the output must look like, and **HOW** to compute it — conceptually.

Layout, statistics, footnotes… but not executable detail.

+

ADaM metadata

Defines **WHICH** variables exist and how how they were derived.

But not how they map to each cell of this table.

→

The gap

Neither specifies the granular detail needed to bridge specs → R code.

Population, denominator, sort order, tie-breaking, covariance matrix – nobody wrote down anywhere

The Promise : let AI close that gap and draft regulatory-grade rtables / junco / tern / tidytlg code.

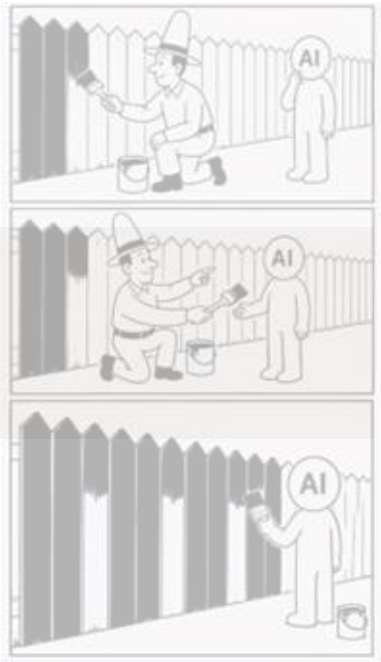
The Catch : *AI is not deterministic.*

THE CATCH

The AI determinism gap

Same mock shell. Same metadata. Very different reasons for landing where they land.

WHO DRAFTS	WHAT FILLS THE GAP	WHERE IT LANDS
Human programmer	Institutional knowledge (SOPs, TLG conventions, prior submissions, review history). Two programmers write different codes and arrive at the same table. The variation is real – but acceptable because it is bounded by standards, conventions, and review.	✓ **Lands on the intended display**
AI, unguided	The model reaches the same fork and chooses. A plausible guess, not an informed decision. Without institutional context, the selection is probabilistic and in the code, both look equally confident.	? **Sometimes right, sometimes not**
AI, guided	Where we need to land. Not a better guess – no guess at all. How we get there is the rest of this talk.	✓ **Lands where we intend, every run**

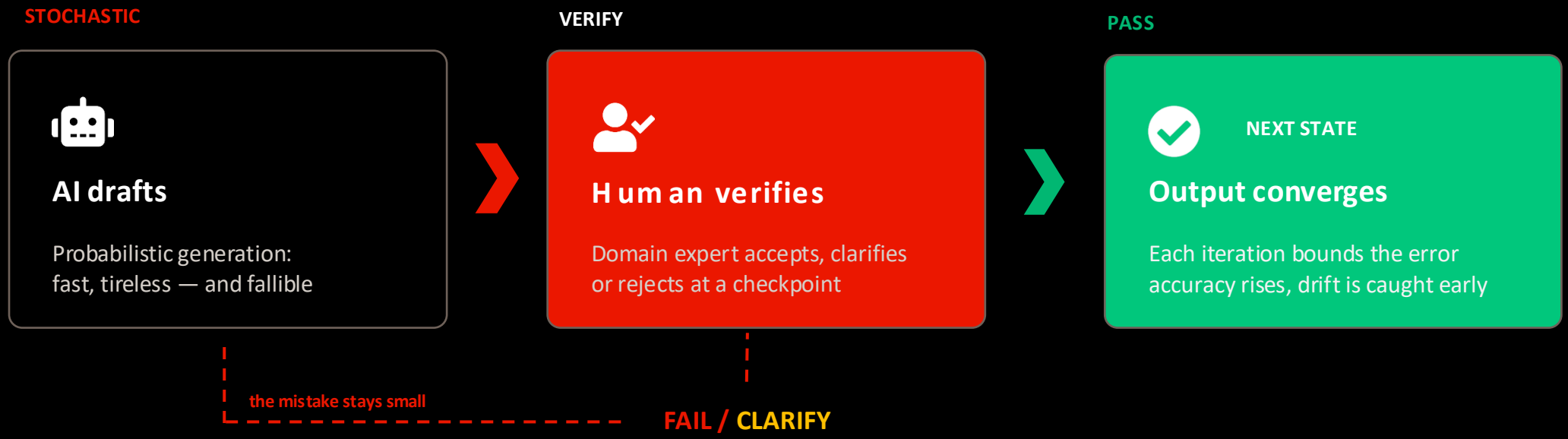


Source: Author unknown

The problem is not that AI may produce different drafts. The problem is whether those differences are constrained by rules that guarantee acceptable outcomes.

THE CORE IDEA

Supervision is the framework that closes the AI determinism gap.



↺ **Don’t validate the model. Validate the end-to-end workflow**
(evidence, gates, outcomes and state transitions).

DESIGN PRINCIPLES

Three AI principles shaped the whole approach

Skepticism explains why we verify. Context defines the evidence. MCP injects it at the right state.

WHY WE VERIFY

Skepticism, curiosity & Human-in-the-Loop (HITL) validation

Healthy doubt as a feature. Every AI proposal is independently validated by the domain expert
YOU are responsible for code quality & accuracy.

WHAT WE VERIFY AGAINST

Context

An LLM without the right context falls back on stale training data. The right context, at the right time –
time – progressive disclosure, is what constrains the answer and ensures correctness.

HOW EVIDENCE ENTERS

Model Context Protocol (MCP) / SKILLS

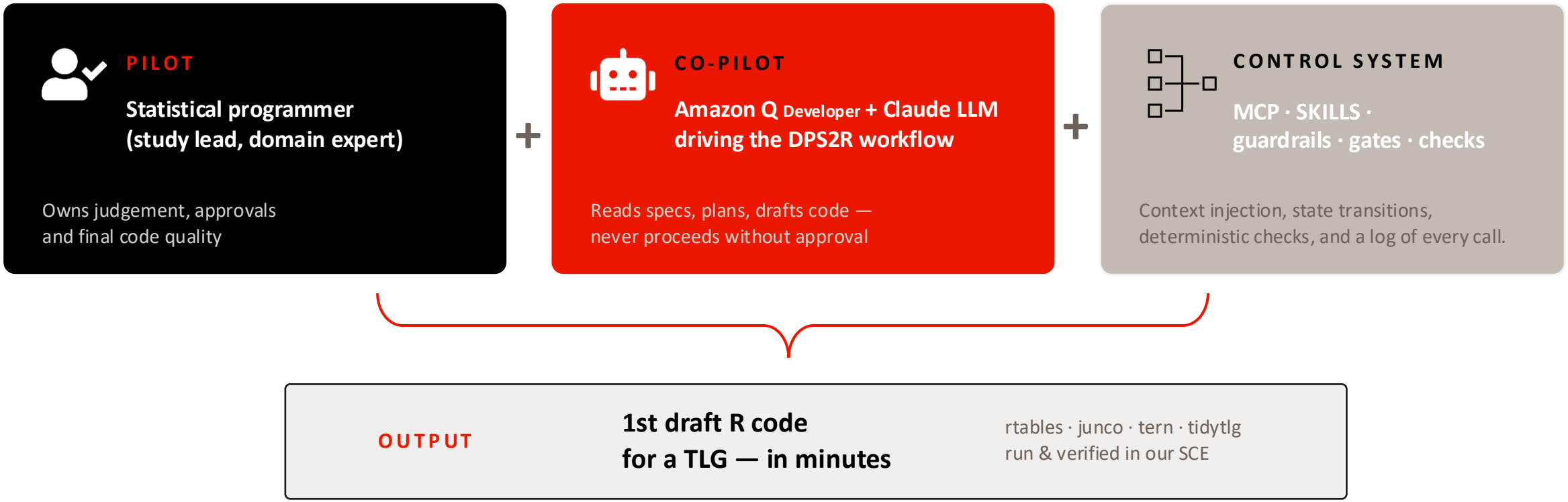
The "USB-C for AI": a standard protocol connecting agents to study metadata, code libraries and J&J guidance. Those
guidance. Those tools deliver phase-specific evidence without letting the model invent capability or facts.
facts.
SKILLS: Eliminates repeated long prompts, keeps instructions out of context until needed, enables consistency and token
consistency and token efficiency.

Take-home from upskilling: the LLM is a "child prodigy willing to hallucinate" - you are the domain expert; AI is your co-pilot.

THE TOOL

DPS2R — a cockpit, not an autopilot

The pilot supplies judgement. The co-pilot supplies generation. The control system governs progression.

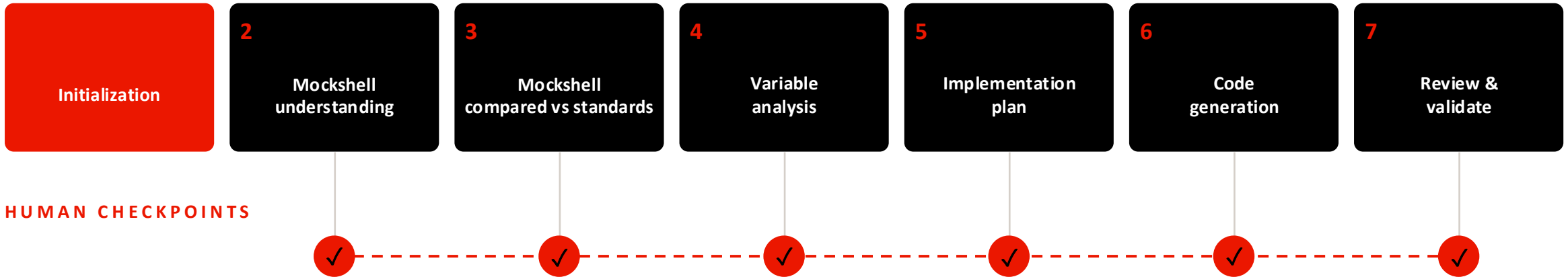


A cockpit gives you instruments, automation, a lot of power -
but it does not fly you somewhere you did not choose, and the pilot never leaves the seat.

THE WORKFLOW

7 phases, 6 human checkpoints

A state-machine: no phase starts without the human confirming the previous one.



@pass

confirm & continue — triggers DPS2R memory updates

@clarify

pause, discuss, request minor adjustments

@fail

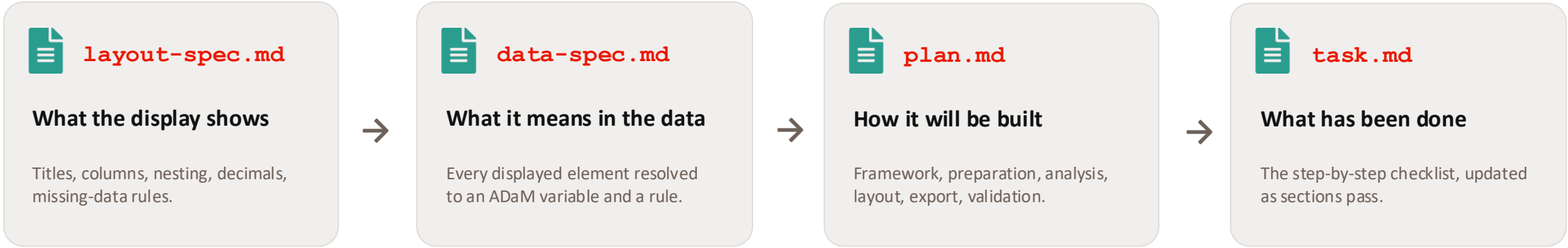
major issue — restart from a chosen phase

Plus, a deterministic safety net before checkpoint 6: syntax check · function attribution · namespace verification — with a full audit trail of every tool call (user, session, timestamp).

WHY IT WORKS

Spec-Driven Development (SDD): artifacts before code

The workflow writes four intermediate artifacts — Each one is approved at a gate, and each is the input the next phase is checked against



Three SDD goals:

1. **Clarification** — resolve ambiguities in existing specs before any code exists
2. **Elaboration** — extract the implementation detail the specs left implicit
3. **Decoupling** — separate the WHAT (specs) from the HOW (code)

A wrong idea about the layout is caught at the layout gate - not three hours later, inside two hundred lines of code you now have to have to throw away…

First, make the output explicit.

MOCKSHELL

Adverse events by SOC and preferred term

Safety population · subjects with ≥1 treatment-emergent event, n (%)

SYSTEM ORGAN CLASS / PREFERRED TERM	TREATMENT A	TREATMENT B
SYSTEM ORGAN CLASS	n (%)	n (%)
Preferred term	n (%)	n (%)
Preferred term	n (%)	n (%)
SYSTEM ORGAN CLASS	n (%)	n (%)

THE DISPLAY SHOWS

APPROVED INTERPRETATION

layout-spec.md

titles + footnotes

column and treatment hierarchy

rows, statistics and indentation

ordering, precision and pagination

missing-value behavior

The programmer approves the interpretation before data choices begin.

Trace every displayed cell back to its data contract.

DATA-SPEC MAPPING

Each shell component names its ADaM contract.

The data-spec records what every displayed element resolves to, before any R is written.

SHELL COMPONENT	WHAT THE DATA-SPEC FIXES	ADaM LINKAGE
Treatment columns	assignment and column order	`ADSL · TRT01A`
SYSTEM ORGAN CLASS rows	first grouping level	`ADAE · AEBODSYS`
Preferred term rows	second level, nested in SOC	`ADAE · AEDECOD`
n (%) cell value	each subject counted once per group	`ADAE · USUBJID`
“Safety population” title	who is eligible to be counted	`ADSL · SAFFL`
“Treatment-emergent” footnote	which records qualify	`ADAE · TRTEMFL`

THE LINKAGE shell component → data-spec entry → ADaM variable

data-spec.md

BEFORE CODE, APPROVE:

WHO COUNTS

Safety population + denominator

ADSL

WHICH RECORDS COUNT

Treatment-emergent records grouped SOC → PT

ADAE

HOW EACH SUBJECT COUNTS

filter · deduplicate · count · sort

PROGRAMMER GATE

This mapping is approved before R code is generated.

IMPLEMENTATION (*plan.md & tasks.md)*

Approved specifications become a plan, then a controlled draft.

plan.md

framework
data preparation
analysis + layout
formatting + export
validation

draft-output.R

Written section by section, so a failure is caught inside one section rather than one script)

SETUP	`library(tern); library(rtables)`
PREPARE	`adsl_pop <- filter(adsl, SAFFL == "Y")`
ANALYZE	`adae_te  <- filter(adae, TRTEMFL == "Y")`
LAYOUT	`lyt <- basic_table() |> split_rows_by(...)`
EXPORT	`export_as_rtf(result, ...)`

DETERMINISTIC CHECKS

PARSE

R syntax : Does the section run at all?

ATTRIBUTE

function → package : Does this function belong to that package?

VERIFY

namespace exists : Does it resolve inside our J&J container?

tasks.md
CONTROL LOOP **Pass → record progress in tasks.md and continue. Fail → stop and return to the programmer.**

LIVE DEMO

DPS2R

One shell. One prompt. One working first draft.

THE LESSON

When results wobbled, the answer was more supervision

Evaluated by 9 experienced programmers · 329 TLG attempts · 9 completed studies. Tables were hardest — the fix wasn't a bigger model.

Evaluated version

2

HITL checkpoints

Accuracy rose with every iteration;
solid on listings & graphs,
tables needed many rounds

→

Production version

6

HITL checkpoints + 7-phase
state machine

Supervision scaled up where
complexity demanded it

The dial goes both ways. As context matures (JJCS TLG library, R adoption in our SCE), fewer checkpoints may suffice. Supervision is a **tunable risk control** — you turn it up when outputs are complex or context is thin, and relax it as evidence accumulates. That is exactly how validated processes are supposed to evolve.

TRUST & GOVERNANCE

AI drafts. Existing validation stays untouched.

DPS2R ends where the output development ends — everything downstream is unchanged.

Ring-fenced

Amazon Q Developer + Claude run inside J&J's enterprise no data leakage to external AI systems.

Same promotion path

Self-checked code → SCE → risk-based validation → production. Failures demote back to dev for further DPS2R iterations.

Audit-ready

Every tool call logged (user, session, parameters, timestamp); spec artifacts document each decision.

Serendipity: upskilling

Reviewing draft rtables/junco code at each checkpoint is hands-on R training - an unexpected benefit for programmers new to the stack.

TAKEAWAYS

The human is the pilot. Keep it that way.

1. **AI is probabilistic — your workflow doesn't have to be**

   Checkpointed iteration turns a non-deterministic generator into a convergent, auditable process.

2. **Supervised is the sweet spot for regulated outputs**

   Approval gates, spec artifacts, audit trails, deterministic validations — governance by design, not afterthought.

3. **Iteration is the framework that closes the gap**

   Accuracy rises every round because a domain expert steers every round. The end-to-end workflow is what you validate.