# SM07: Metadata-driven automatic ADaM R code generation

Thiago Figueiredo – Merck KGaA, Darmstadt, Germany

PHUSE EU Connect 2023, 08.11.2023

MERCK

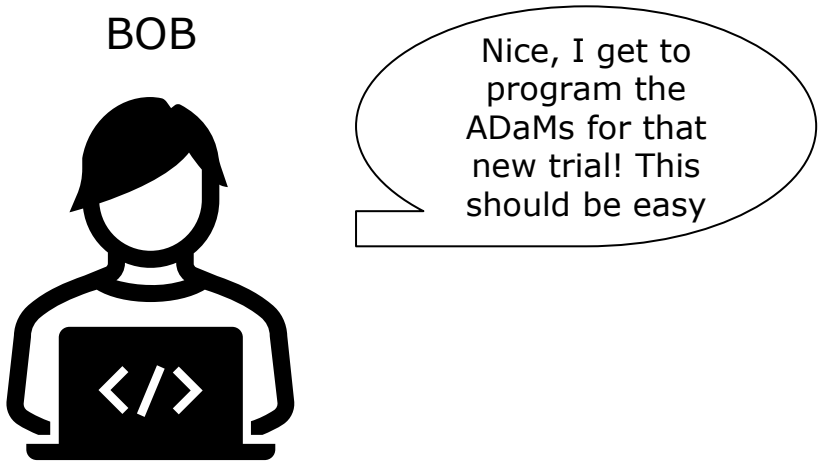
# Overview of the presentation

The views expressed in this talk are those of the speaker and not necessarily those of Merck KGaA.

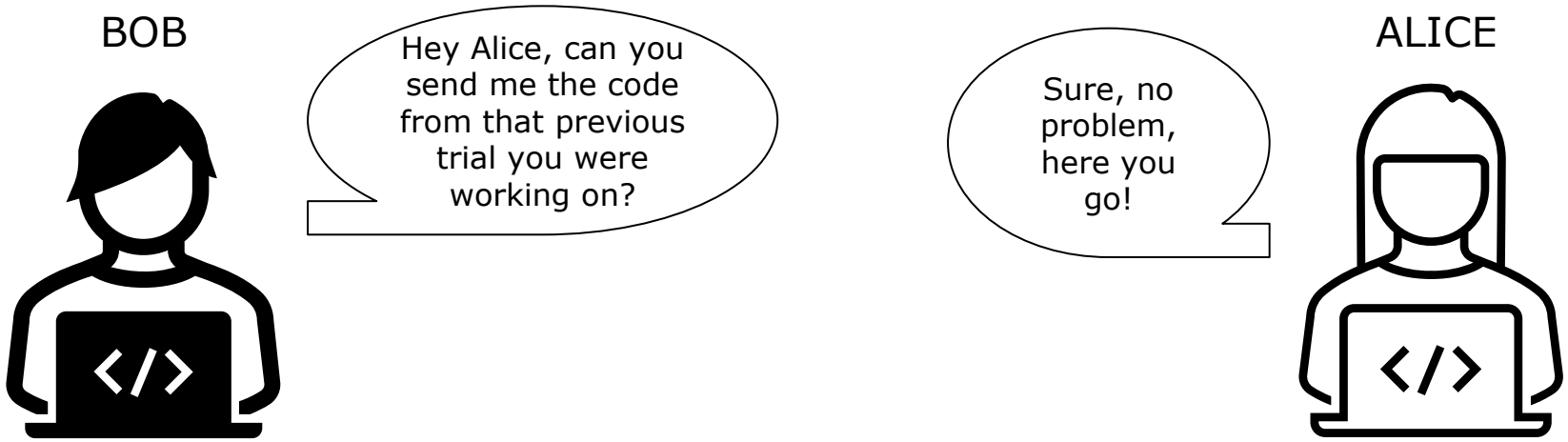
# Overview of the presentation

1. A short story about ADaM programming
2. Why you should care about Ctrl C+ Ctrl V
3. Leveraging metadata for ADaM R Code generation in three steps
4. Benefits for trial programmers
5. Current limitations
6. Importance of MDR and integration into the clinical data pipeline
7. Future development
8. Key Takeaways

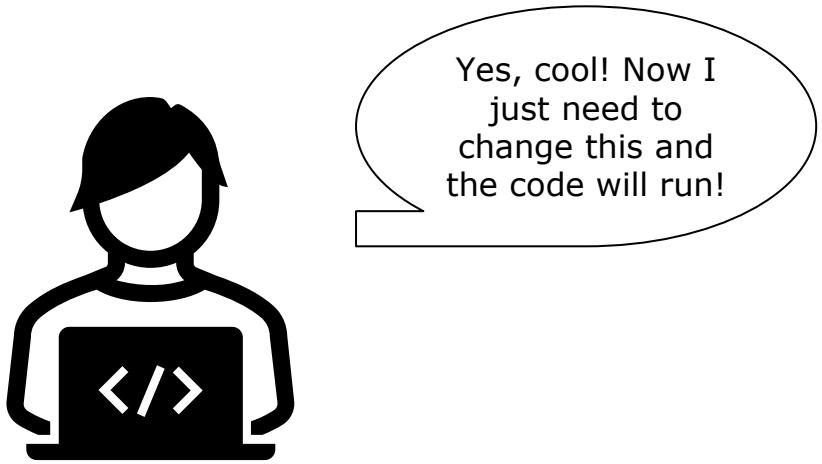
# ADaM Programming: A story based on true events



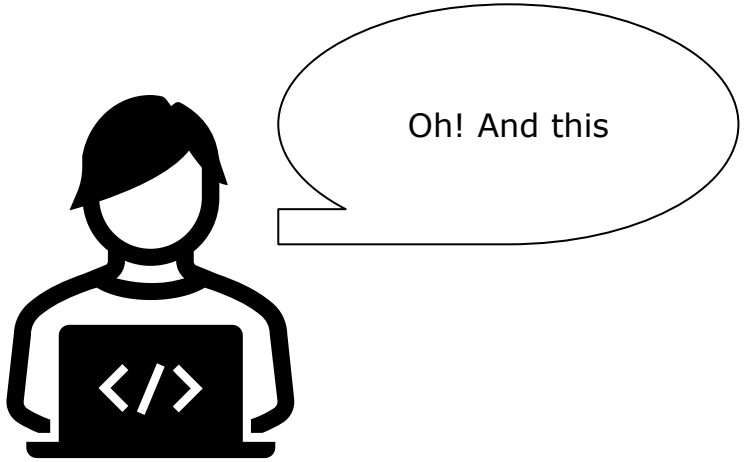
# ADaM Programming: A story based on true events



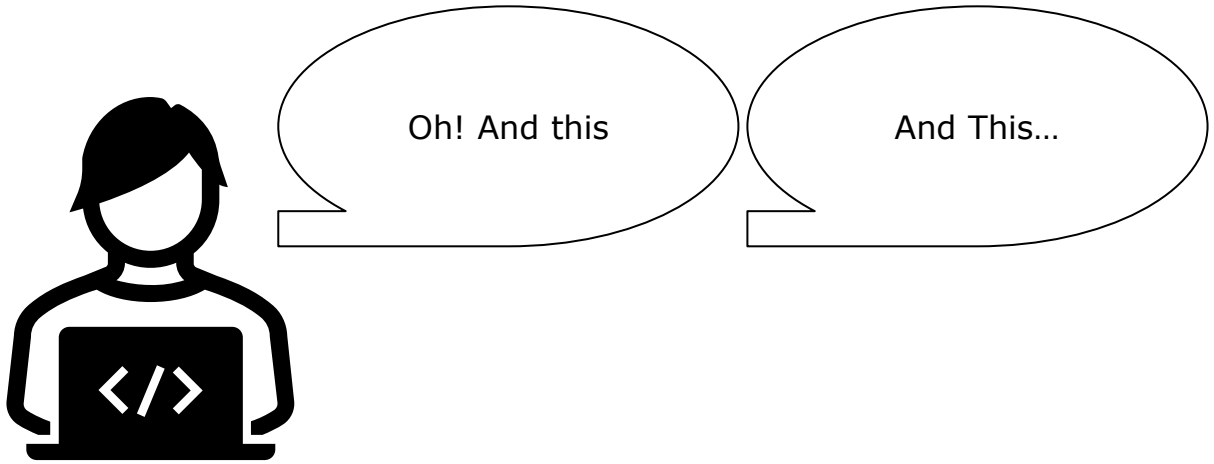
# ADaM Programming: A story based on true events



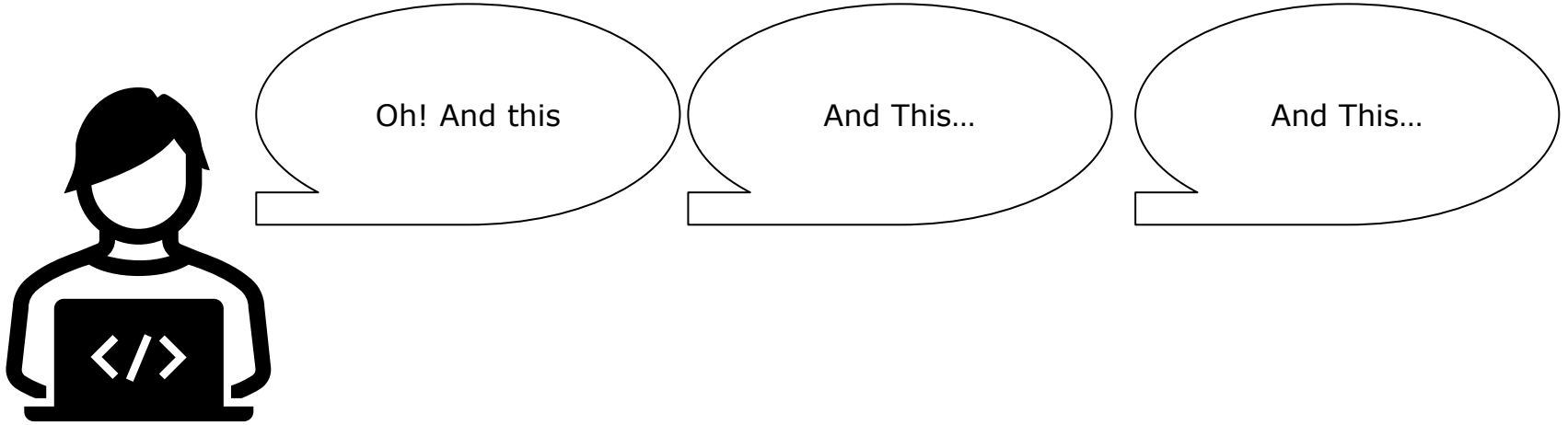
# ADaM Programming: A story based on true events



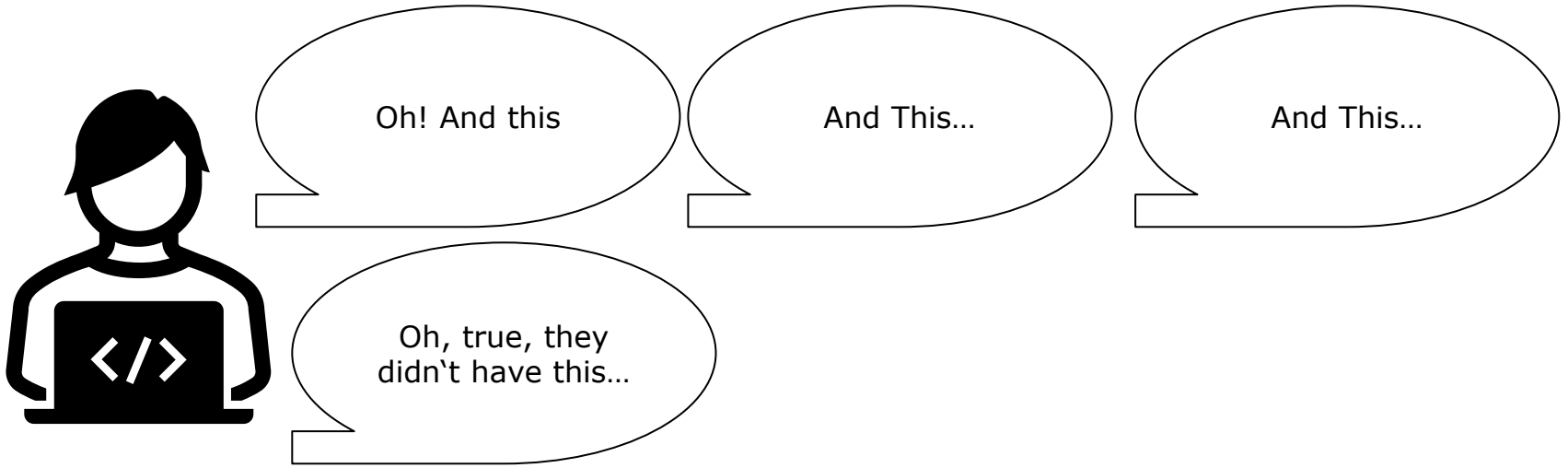
# ADaM Programming: A story based on true events



# ADaM Programming: A story based on true events

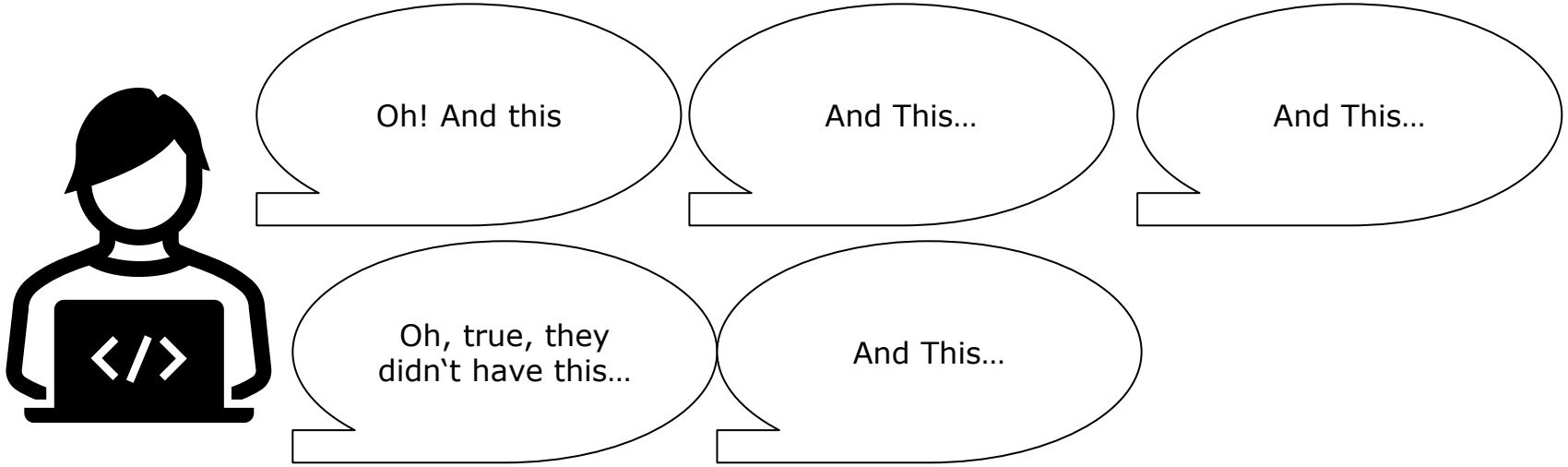


# ADaM Programming: A story based on true events



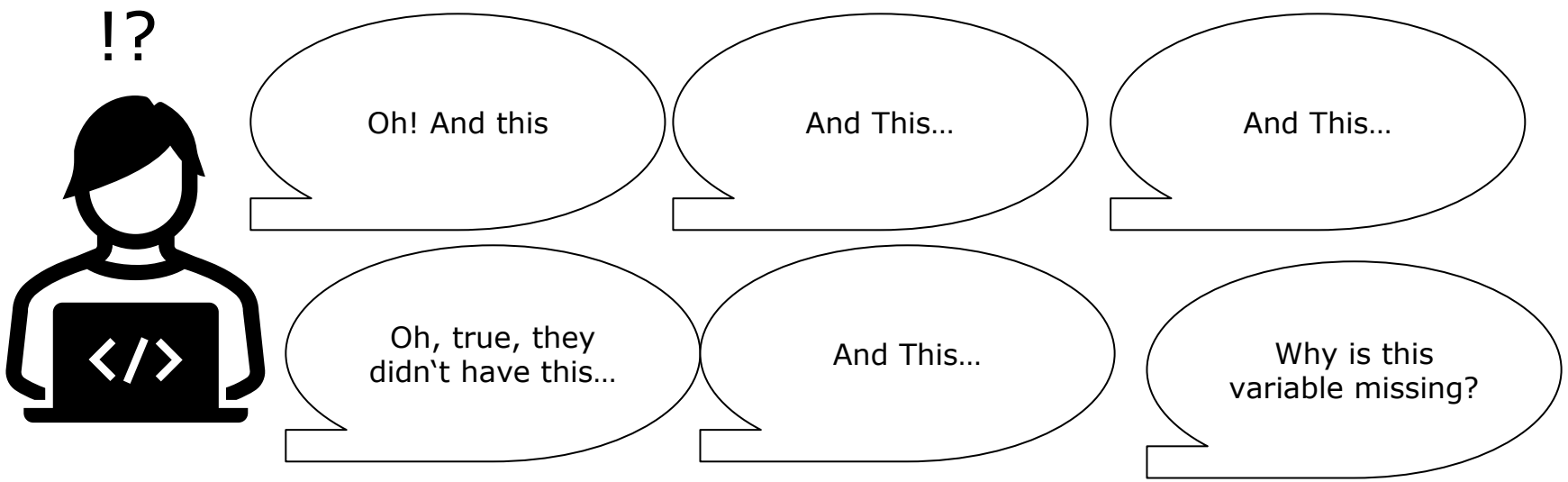
# ADaM Programming: A story based on true events

Deadline now approaching at full speed…



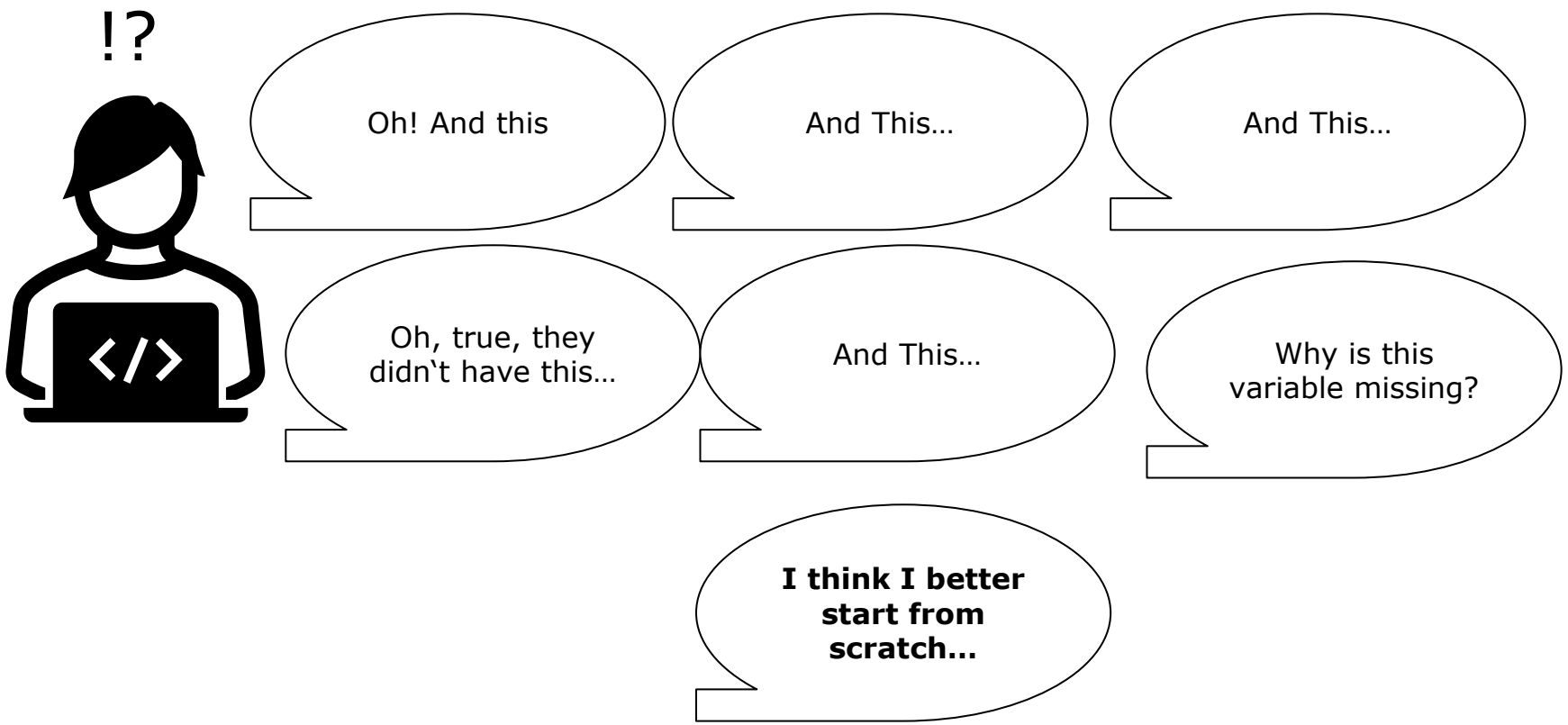
# ADaM Programming: A story based on true events

Deadline now approaching at full speed…



# ADaM Programming: A story based on true events

Deadline now approaching at full speed…



# Copy + Paste is bad and this is why you should care

- Copied code **may contain irrelevant or misleading information**
- Copied code from different trials **may omit relevant information** to the current trial
- Copied code **may be outdated**
- Changes in copied code are **not traceable**

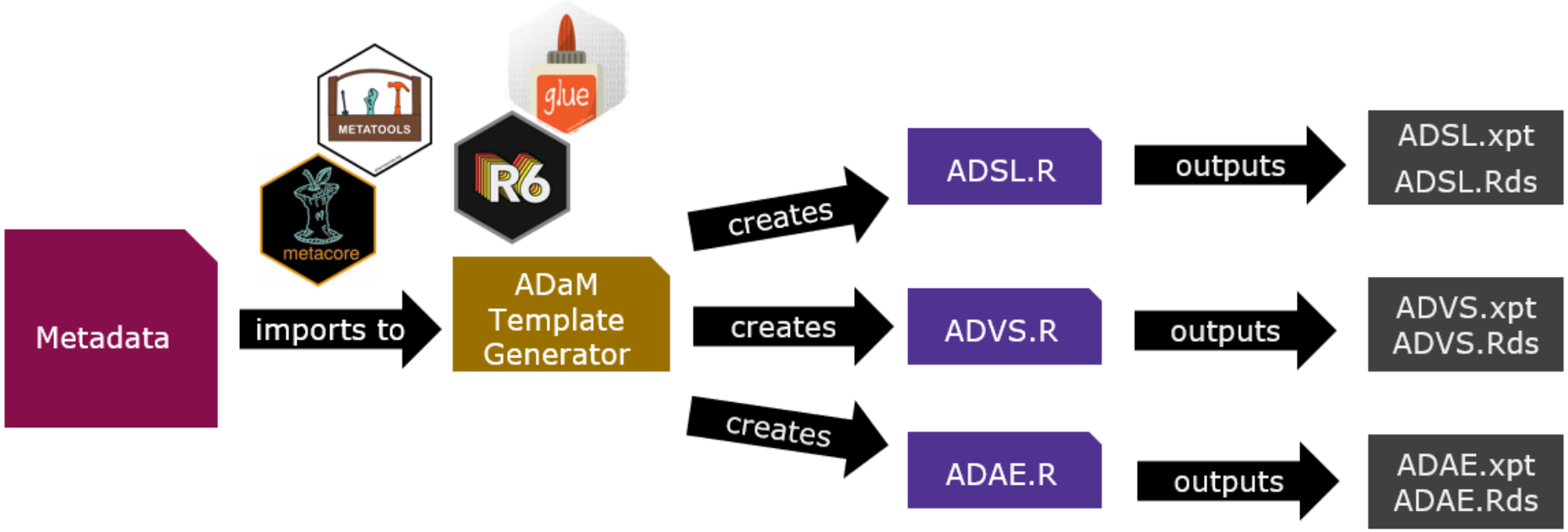
# But wait, there was something… Of course, metadata!

| Key | Level | Dataset | Variable | Label | Include | Predecessor | Computation Expression | Code Type | Computation Type |
|---|---|---|---|---|---|---|---|---|---|
| rc_key | rc_level | ds_name | va_name | va_label | rc_incyn | va_predecessorname | va_methodexpression | va_methodcontext | va_methodtype |
| ADSL.STUDYID | VA | ADSL | STUDYID | Study Identifier | Y | DM.STUDYID | | | |
| ADSL.USUBJID | VA | ADSL | USUBJID | Unique Subject Identifier | Y | DM.USUBJID | | | |
| ADSL.SUBJID | VA | ADSL | SUBJID | Subject Identifier for the Study | Y | DM.SUBJID | | | |
| ADSL.SITEID | VA | ADSL | SITEID | Study Site Identifier | Y | DM.SITEID | | | |
| ADSL.REGION1 | VA | ADSL | REGION1 | Geographic Region 1 | Y | | | | |
| ADSL.REGION1N | VA | ADSL | REGION1N | Geographic Region 1 (N) | Y | | | | |
| ADSL.BRTHDTC | VA | ADSL | BRTHDTC | Date/Time of Birth | Y | DM.BRTHDTC | | | |
| ADSL.AGE | VA | ADSL | AGE | Age | Y | DM.AGE | | | |

... datasets **variables** codelists dictionaries methods documents arms adamig_variables ...

- Information is **trial- and task-specific**, including methods and derivations
- ADaM Metadata contain **complete information** about datasets included in the task
- Information **must be up-to-date for regulatory compliance**

# Leveraging structured metadata provides an advantage for programming




Metadata are read into the metacore object and used by the template generator module to produce executable code templates

# Metacore package provides a relational structure for storing spec information

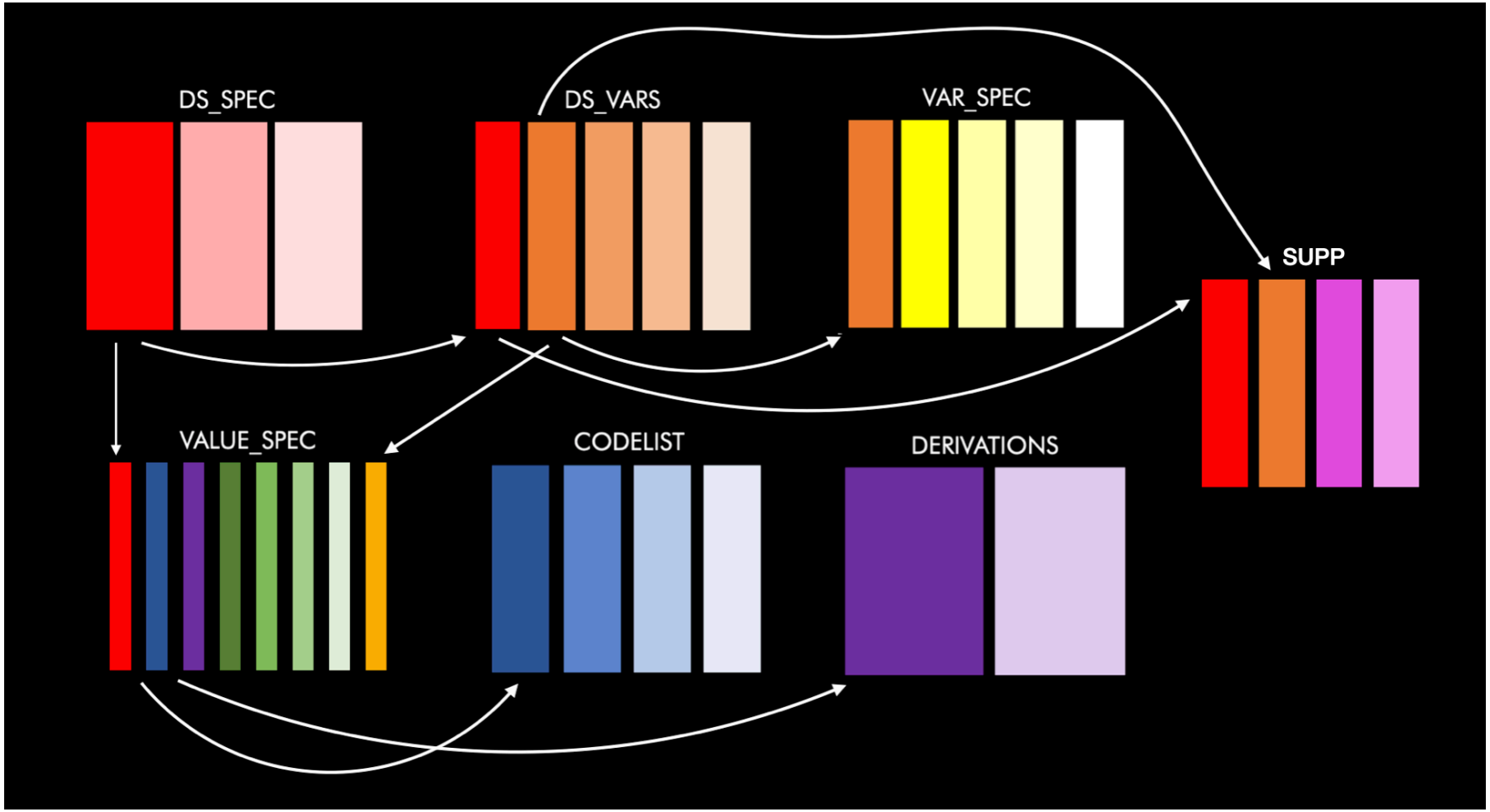



Image from the [metacore github page]

# ADaM R Code generation in three steps
## Step 1: Read Metadata from … wherever they are

- **Metacore** accepts the p21 specs as default, but provides help functions to read custom specs formats
- The package contains functions to read xml files or excel sheets and already provides **internal consistency checks**. It also contains help functions to filter the metacore object for a given dataset
- Information stored in other form (e.g. on an SQL Server) can also be **read separately and saved as metacore object**
- Metacore objects are **R6 objects**
- The **metatools** package contains **additional functions** for using the metacore object to directly create the output dataset

# ADaM R Code generation in three steps
## Step 2: generate single template with the template generator module

```
# Create object ------------------------------------------------------------------

rfile←CodeGen$new(paste0(outfolder,ds_pars$dataset_name,".R"))

rfile$comment("Author: <?ds_pars$author?>")
rfile$comment("Created: <?Sys.time()?>")
rfile$comment("Task: <?ds_pars$dataset_name?> -- Template Code Generation")
rfile$comment("Remarks: ")

rfile$bl()

rfile$l("rm(list=ls())")

rfile$bl()

rfile$section("load libraries")

rfile$bl()

# load libraries and source other files

rfile$addLines(c("library(dplyr)","library(lubridate)","library(admiral)","library(haven)","library(openxlsx)"))

rfile$bl()

rfile$comment("load codelists from the specs")

rfile$bl()

rfile$addChunk("relevant_domains←c('ae','cm','da','dm','ds','ec','eg','ex','ie','is'
'lb','pc','pr','qs','rs','sc','sv','tr',
ts','tu','vs','suppdm','suppec','suppae','suppsc')")

rfile$bl()

rfile$addLines("sdtm←load_sdtm(sdtm_folder,relevant_domains)")
rfile$bl()

rfile$addChunk("sdtm←lapply(sdtm,as.data.frame)")

 ...|

rfile$section("Create Predecessor Variables")

rfile$bl()

for (v in predecessor_variables)
{
  rfile$comment("Create <?this_val_spec[this_val_spec$variable_key== v,]][['origin']]?> Variable -- <?v?>")
  rfile$bl()
  this variable←this val spec[this val spec$variable key==v,]][['variable']]
```

- One template program can **dynamically generate code for all dataset** from their respective metadata
- Dynamically generated lines of code are written to an R file using the **Codegen** class (R6)
- Fixed strings and environment variables are integrated using the **glue package**
- **Single lines of code, as well as code chunks** can be added using the Codegen

# ADaM R Code generation in three steps
## Step 2: generate single template with the template generator module

| Key | Level | Dataset | Variable | Label | Where | Include | Key Order | Origin | Predecessor | Computation Type | Comment |
|---|---|---|---|---|---|---|---|---|---|---|---|
| rc_key | rc_level | ds_name | va_name | va_label | vl_where | rc_incyn | va_keyorder | va_origin | va_predecessorname | va_methodtype | va_comment |
| ADAE.DLT | VA | ADAE | DLT | Dose Limiting Toxicity | | Y | | Predecessor | SUPPAE.QVAL@QNAM="DLT" | | Set to 'Y' if the investigator believes the record is a dose-limiting toxicity (DLT) |
| ADAE.SMCDLT | VA | ADAE | SMCDLT | Is this event a DLT according to SMC | | Y | | Predecessor | SUPPAE.QVAL@QNAM="SMCDLT" | | Set to 'Y' if the SMC determined the record is a dose-limiting toxicity (DLT) |

```r
# Create Predecessor Variable -- ADAE.DLT

# !Check

ADAE <- ADAE %>%
  left_join(sdtm$SUPPAE %>%
    filter( ... ) %>% # QNAM==DLT
    select(STUDYID, USUBJID, AESEQ, QVAL),
  by = c("STUDYID", "USUBJID", "AESEQ")
  ) # Check Keys
names(ADAE)[names(ADAE) == "QVAL"] <- "DLT"
```

```r
# Create Predecessor Variable -- ADAE.SMCDLT

# !Check

ADAE <- ADAE %>%
  left_join(sdtm$SUPPAE %>%
    filter( ... ) %>% # QNAM==SMCDLT
    select(STUDYID, USUBJID, AESEQ, QVAL),
  by = c("STUDYID", "USUBJID", "AESEQ")
  ) # Check Keys
names(ADAE)[names(ADAE) == "QVAL"] <- "SMCDLT"
```

- Once the metadata are available in the session, they can be used to **generate code templates**
- Simple variables can be assigned directly, more **complex derivations may contain placeholders** to be checked by the trial programmer

# ADaM R Code generation in three steps
## Step 2: generate single template with the template generator module

```r
# Create Predecessor Variable -- ADAE.SMCDLT

# !Check

ADAE ← ADAE %>%
  left_join(sdtm$SUPPAE %>%
    filter( ... ) %>% # QNAM==SMCDLT
    select(STUDYID, USUBJID, AESEQ, QVAL),
  by = c("STUDYID", "USUBJID", "AESEQ")
  ) # Check Keys
names(ADAE)[names(ADAE) == "QVAL"] ← "SMCDLT"

# Joining ADSL (reuse) Variables

adsl_reuse ← c("STUDYID", "USUBJID", "SUBJID", "SEX", "ARACE", "FASFL", "SAFFL", "DLTFL", "PKFL",

ADAE ← ADSL["adsl_reuse"] %>%
  left_join(ADAE, by = "USUBJID")

# Create Derived Variables ————————————————————————————————

# Create Derived Variable -- ADAE._ACNLST

# Create Derived Variable -- ADAE._ACNOTH

# Create Derived Variable -- ADAE._AOUT

# Create Derived Variable -- ADAE._ASERLST

# Create Derived Variable -- ADAE._WORSEFL

# Create Derived Variable -- ADAE.AACN

# Create Derived Variable -- ADAE.AACNOTH
```

- Once the metadata are available in the session, they can be used to **generate code templates**
- Simple variables can be assigned directly, more **complex derivations may contain placeholders** to be checked by the trial programmer
- **All necessary variables for a given dataset are made available** in the template, based on the corresponding metadata

# ADaM R Code generation in three steps
## **Step 3: generate all datasets in batch**

```r
# Author: Thiago
# Created: 2023-08-26 10:22:05
# Task: ADAE -- Template Code Generation
# Remarks:

rm(list = ls())

# load libraries ---------------------------------------------------------

library(dplyr)
library(lubridate)
library(admiral)
library(haven)

# load sdtm files

relevant_domains <- c(
  "ae", "cm", "da", "dm", "ds", "ec", "eg", "ex", "ie", "is",
  "lb", "pc", "pr", "qs", "rs", "sc", "sv", "tr",
  "ts", "tu", "vs", "suppdm", "suppec", "suppae", "suppsc"
)

sdtm <- load_sdtm(sdtm_folder, relevant_domains)

sdtm <- lapply(sdtm, as.data.frame)

names(sdtm) <- toupper(names(sdtm))

# Create output dataset

n_rows <- nrow(sdtm$AE)

ADAE <- tibble(.rows = n_rows)

# Start programming variables --------------------------------------------

# Create Predecessor Variables -------------------------------------------

# Create Predecessor Variable -- ADAE.AEACN

ADAE$AEACN <- sdtm$AE$AEACN

# Create Predecessor Variable -- ADAE.AEACNOTH

ADAE$AEACNOTH <- sdtm$AE$AEACNOTH

# Create Predecessor Variable -- ADAE.AEBODSYS

ADAE$AEBODSYS <- sdtm$AE$AEBODSYS
```

- The generated template **contains all variables, as well as general setup code** (e.g. package loading, environment setup, etc)
- The **same procedure can be deployed to produce all code templates** for a given task
- Templates are already formatted using the **styler package**

# What’s in it for the trial programmer

- Code templates are generated based on **task-specific metadata**
- Code templates contain **all variables needed for the datasets**, based on available metadata
- All „**predecessor**“ variables and virtually all **assigned** variables are readily available and require little to no changes
- This usually amounts to **60-80% of all variables being directly available**, depending on the dataset
- **More complex derivations** may also be available, drawing from standard methods from the „derivation“ table (e.g. using admiral functions)

# Current limitations

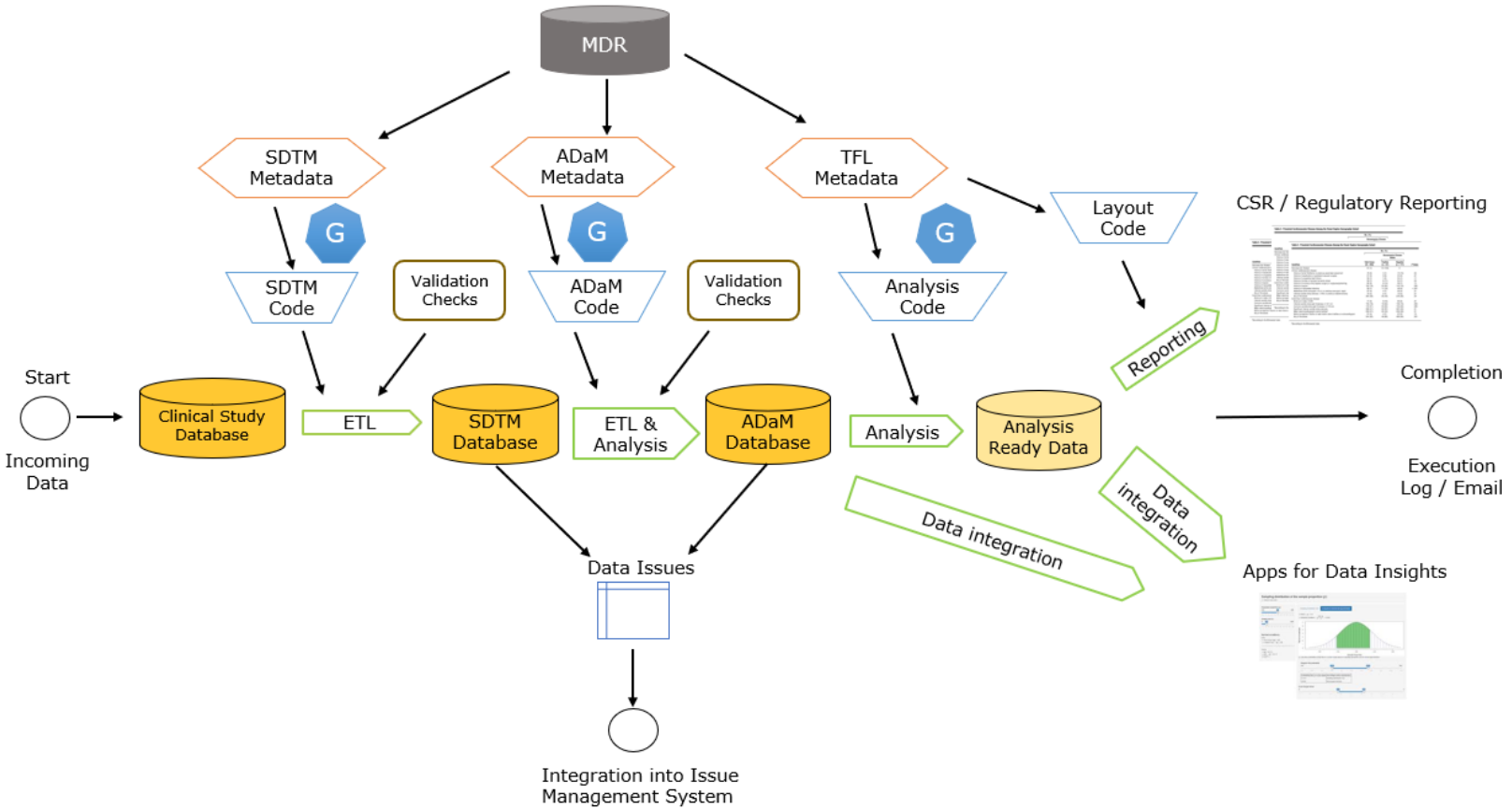
Some technical aspects are currently being optimized based on initial user feedback:

- Variables are included in the template according to a **fixed order**, based on variable origin: **Predecessors, Derived, Assigned**
- Each variable has its own derivation (**no clustering of related variables** like TRT01P & TRT01PN)
- Template module is **specific to ADaM**
- **Complex derivations** are currently **not yet included**

# Why it is a good idea to implement an MDR

- Information from multiple trials may be available in a single place and allow for **effective comparison and search**
- Central code **templates** for each dataset **can be improved over time** and made available to all trials
- This stimulates **coding best-practices** (e.g. commenting guidelines, use of certain packages, etc)
- The same **module may be recycled** to produce code templates **for different standards** (e.g. SDTM, TLF)
- Integration into the clinical landscape may allow end-to-end **linking of metadata** from raw data to TLF

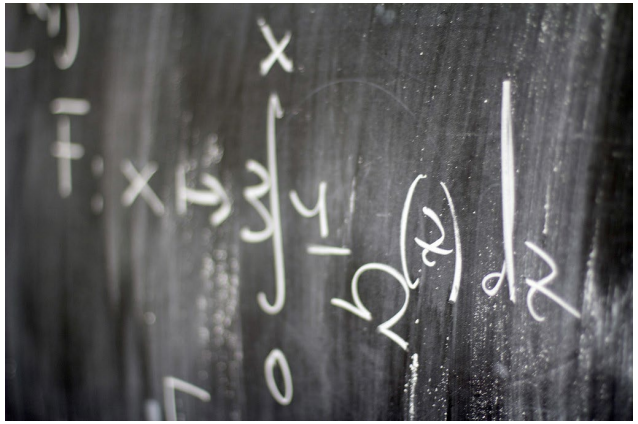
# Automation potential of integrating the tool into the clinical data pipeline



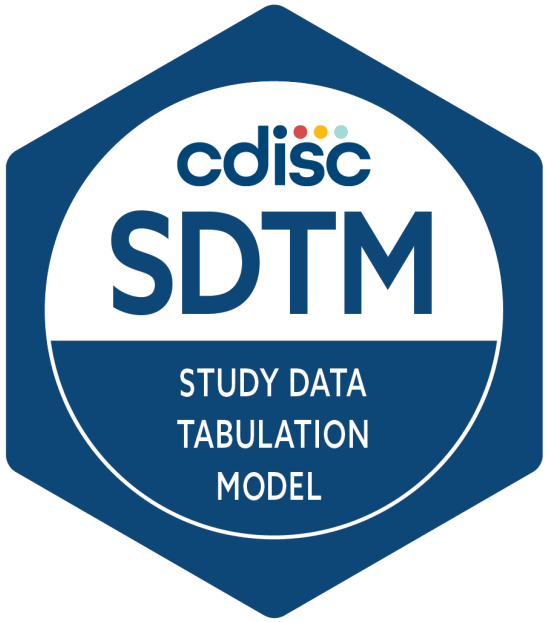
# Future development

Improve ADaM Tool

Include additional derivation variables



Extend to SDTM



# Key Takeaways

- Metadata can help us **break free from the copy+paste cycle**, but there is no free lunch
- They allow us to **use task-specific information** to generate R code automatically
- The **metacore** and **metatools** package provide a structure to store and manipulate metadata in R
- The template module leverages CDISC structure to **automate 60%-80% of dataset programming**
- **All datasets can be produced with the same „Template Generator"** module
- The framework is **flexible and can be adapted** to suit the end-user's needs

# Recommended reading

- https://github.com/r-lib/R6 (R6 package)
- https://glue.tidyverse.org/ (glue package)
- https://atorus-research.github.io/metacore/ (metacore package)
- https://pharmaverse.github.io/metatools/ (metatools package)
- https://github.com/r-lib/styler (styler package for code formatting)
- https://github.com/pharmaverse/admiral (Package for standard ADaM derivations)

# questions?

**Thiago Figueiredo**
thiago.figueiredo@merckgroup.com

**Waldemar Miller**
waldemar.miller@merckgroup.com