

Why I can't see myself coding that much anymore

PHUSE EU Connect 2025
2025.11.17

Angelo Wibbeler
Daiichi-Sankyo Europe GmbH

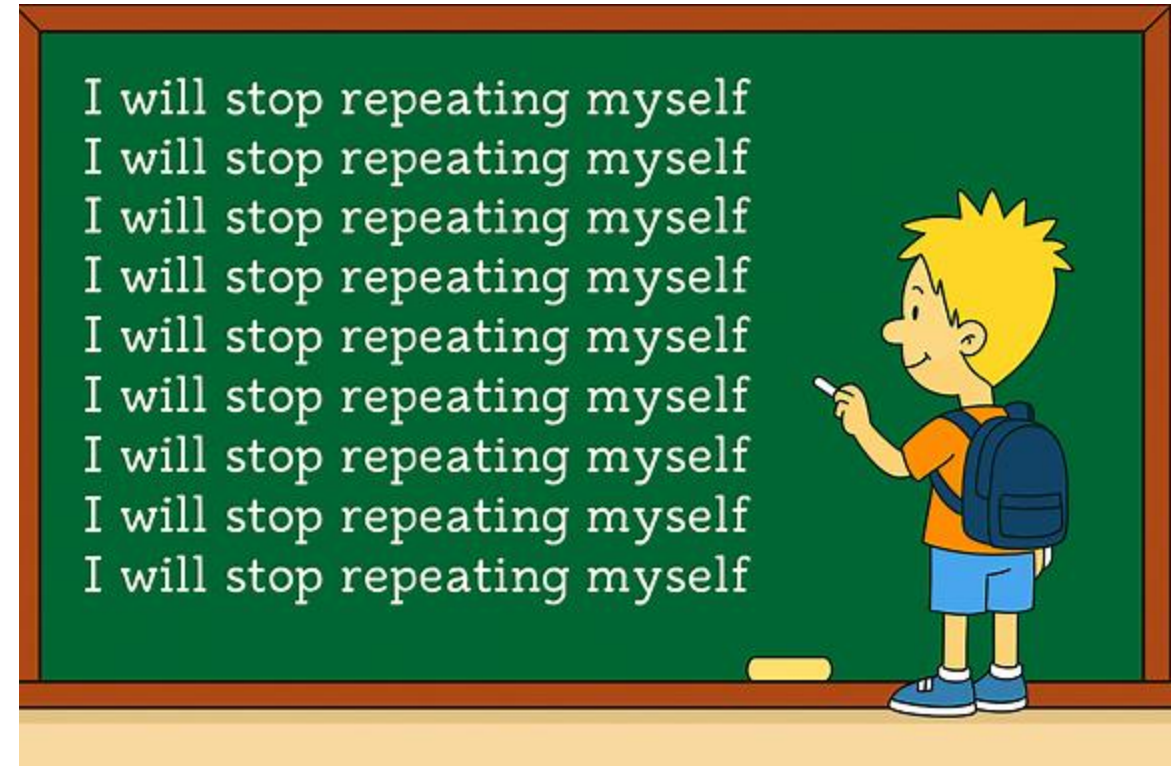
Don't repeat yourself (DRY) is about more than just code

Why is DRY important?

- Improves Consistency
- Reduces Maintenance Overhead
- Boosts Clarity & Focus
- Fundamental principles in professional programming
→ We should listen to decade old industry best practices

Where it applies apart from code?

- Documentation
 - Processes & Workflows
 - Communication
- ... everywhere



Don't repeat yourself (DRY) is about more than just code

“DRY is about the duplication of knowledge, of intent.”
-David Thomas, Andrew Hunt. The Pragmatic Programmer

“The best code is no code at all.”
-Robert C. Martin

Why is this bad?

- This code holds no new information
 - It is a mere duplication of information from the spec, which is not DRY
- High effort to maintain and error prone

```
data adam.adsl;  
  retain USUBJID ...;  
  keep USUBJID ...;  
  attrib  
    USUBJID length=$20 label="Unique Subject Identifier"  
    ...  
;  
...  
proc sort; by usubjid;  
run;
```

Instead, a single macro handling reusage of metadata should be utilized



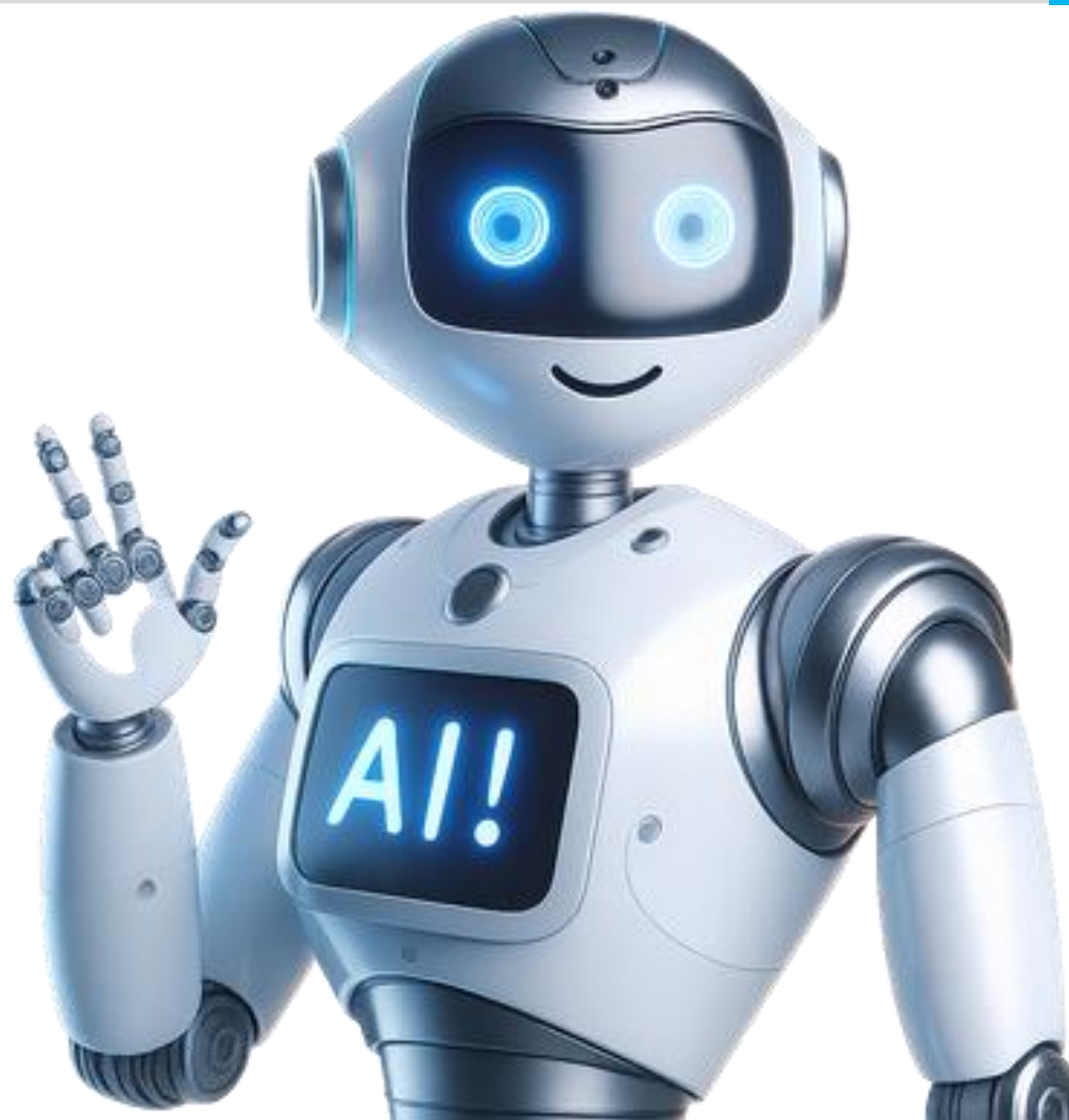
```
%finalize_adam(program = adsl);
```


Daiichi-Sankyo

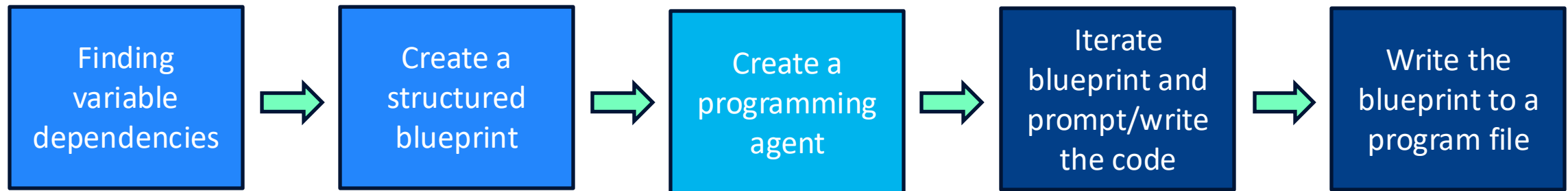
What most companies reuse:

Daiichi-Sankyo

PHUSE EU Connect 2025 – SM06



5 steps to write entire programs with AI



Step 1: Finding variable dependencies

- AI is more effective if we reduce task complexity where possible
- Instead of simply handing over the entire specification, we create structure to guide code generation
- Find dependencies using simple graph searching algorithms (e.g. DFS)
- Dependency levels will ensure automatic code is written in the correct order

Variable	Source / Derivation	Order	Dependency Level
SITESET	ZS.ZSSTRESC where ZS.ZSTESTCD = "SITESET".	5	0
SITESETN	Assign in accordance with the codelist and SITESET value.	6	1
SITEGRP	If SITESETN in (1 2 3) then set to 1, else 2	7	2

Step 2: Create a structured blueprint

- Not all code will be programmed by the AI agent. → Generic sections can be generated automatically like **header** /**end** / or **autocall** sections for fully automated variables
- The remaining variables get their own section (clusters are possible) and will be written by the AI agent

section_name	order	content	Source / Derivation	variables	output_dataset	referenced_vars_datasets
HEADER_	0					{}
autocall_0	1		STUDYID SUBJID SITEID SITESET RFICDT WICDT	autocall_0		{}
IN001FL	3	If IE.IESTRESC = "N" where IN001FL		IN001FL		{}
IN002FL	4	If IE.IESTRESC = "N" where IN002FL		IN002FL		{}
SOSDT	10	Set to numeric SV.SVSTD1SOSDT		SOSDT		{}
FU1VSDT	12	Set to SV.SVSTDTC as num FU1VSDT		FU1VSDT		{}
FU2VSDT	13	Set to SV.SVSTDTC as num FU2VSDT		FU2VSDT		{}
EOSDT	14	Set to numeric max(SV.SV EOSDT		EOSDT		{}
DTHDT	15	Set to numeric DD.DDDTC DTHDT		DTHDT		{}
autocall_1	17		SITESETN IN001FN IN002FN IN003FN IN004FN	autocall_1		{}
SITESPEC	18	Set to ZS.ZSSTRESC where SITESPEC		SITESPEC		{'autocall_0': ['SITEID']}
AGEGR1N	22	if . < AGE < 55 then set to 1 AGEGR1N		AGEGR1N		{'autocall_0': ['AGE']}
autocall_2	32		AGEGR1 AGEGR2 AGEGR3 ASEX	autocall_2		{}
IEFN	33	Set to 1 if all of the inclus IEFN		IEFN		{'autocall_1': ['IN001FN', 'IN002FN', 'IN003FN', 'IN004FN', 'IN005FN', 'IN006FN'], 'autocall_0': ['RFICDT']}
autocall_3	34		IEFL	autocall_3		{}
ENRFL	35	Set to "Y" if part of USUBJI ENRFL		ENRFL		{'autocall_3': ['IEFL'], 'autocall_1': ['COUNTRYN']}
FAS1FL	37	If BASFL = "Y" and not mis: FAS1FL		FAS1FL		{'BASFL': ['BASFL'], 'FU1VSDT': ['FU1VSDT']}
FAS2FL	38	If BASFL = "Y" and not mis: FAS2FL		FAS2FL		{'BASFL': ['BASFL'], 'FU2VSDT': ['FU2VSDT']}
loopref_999	42	{'EOSSTT': 'Part 1:\nIf DS.I EOSSTT DCSREAS DCSREASN DCSREASP		loopref_999		{'autocall_0': ['COUNTRY', 'ACTARM'], 'FU2VSDT': ['FU2VSDT'], 'FU1VSDT': ['FU1VSDT']}
__END__	43					{}

Step 3: Create a programming agent

- Creating simple AI agents became remarkably easy with OpenAI's new SDK for AI agents

```
from agents import Agent
from dotenv import load_dotenv

load_dotenv() # Load environment variables (API key) from .env file

agent = Agent(
    name="SAS_Programming_Agent",
    instructions=context,
    model="gpt-5"
)
```

- The context/instruction for the agent will be a general prompt explaining the purpose of the Agent

```
context = (
    f"You are a {programming_language} programming agent for {programming_task}. We are programming {dataset_name}."
    f"You will be asked to generate {programming_language} code based on a specification."
    "Referenced variables in the derivation instruction are written in uppercase and unquoted."
    "Two upcase words connected by a dot indicate a variable and dataset reference, "
    "whereby the word before the dot represents the dataset name."
    "For variables that have no dataset prefix, a list of datasets where to find the variable is provided."
    f"Key-variables to link datasets together are: {keyvars} "
```

...

Step 4: Iterate blueprint and prompt/write the code

- Iterate over the blueprint and write either 'fixed' content to a section or ask the AI
- To prompt the agent for code, we simply need to use the Runner class

```
from agents import Runner
import asyncio

for idx, row in blueprint_df.iterrows():
    ...
    if get_code_suggestion_condition(row):
        response = asyncio.run(Runner.run(agent, prompt))
        blueprint_df.at[idx, 'content'] = response.final_output
    ...
```

- The prompt for each section will be created in the same iteration

```
# Placeholder for code suggestion logic
prompt = (
    f"Please write me concise code for the variable/s '{row['variables']}'.\n"
    f"The resulting dataset should be called {row['output_dataset']}. "
    f"If intermediate datasets are needed, use the same name with a numeric suffix like _xx.\n"
)
if not pd.isna(row['Referenced Vars']):
    add_referenced_var_information(prompt)
prompt += f"Derivation instruction:\n{row['Source / Derivation']}\n\n"
```

Step 5: Write the final blueprint to the programming file

- Now simply write the content to the respective programming file
- Add section indicators for a clear structure and backwards compatibility
- An abstract example of the final program is shown.
 - The *BMICATBL* variable is derived from an AI without further modification

```
1  *Add your custom header here;
2  %load_input_datasets(program = adsl);
3  * --- __HEADER__ END ---;
4
5  * --- autocall_0 START ---;
6  %autocall_vars(program = adsl, output=autocall_0, vars = STUDYID SUBJID SITEID ...);
7  * --- autocall_0 END ---;
8
9  *...
10 Placeholder for X additional section
11 ...;
12
13 * --- BMICATBL START ---;
14 proc sort data=VS(where=(VSTESTCD='BMI' and VISITNUM=1))
15     out=BMICATBL_01(keep=USUBJID VSSTRESN) nodupkey;
16     by USUBJID;
17 run;
18
19 data BMICATBL;
20     length BMICATBL $12;
21     merge SOSDT(in=a) BMICATBL_01(in=b);
22     by USUBJID;
23     if a;
24     if not missing(SOSDT) then do;
25         if not missing(VSSTRESN) and VSSTRESN<18.5 then BMICATBL="< 18.5";
26         else if 18.5<=VSSTRESN<25 then BMICATBL="[18.5, 25)";
27         else if 25<=VSSTRESN<30 then BMICATBL="[25, 30)";
28         else if 30<=VSSTRESN<35 then BMICATBL="[30, 35)";
29         else if 35<=VSSTRESN<40 then BMICATBL="[35, 40)";
30         else if 40<=VSSTRESN then BMICATBL=">= 40";
31     end;
32     keep USUBJID BMICATBL;
33 run;
34 * --- BMICATBL END ---;
35
36 *...
37 Placeholder for X additional section
38 ...;
39
40 * --- __END__ START ---;
41 %finalize_adam(program = adsl);
42 *Add your custom end section here;
43 * --- __END__ END ---;
```

We ran a modified ADSL spec of a legacy study ADSL output

- Almost no syntax issues:

```
data=FAMH(keep=USUBJID FASTRESC where=(FATESTCD="RISKCL"))
```

- Most issues related to formats/informats (or the spec)

```
input(SVSTDTC, anydtdte.);
```

- 668 lines of generated code vs 400 in original code (after alignment)
- Decoupled code for each variable felt comfortable (no risk of side effects), however, having related variables separated felt rather unnatural
- 1,5h as *time to match*, with an estimation of 4-5h for manual coding



Benefits:

- Timesaving through rapid ADaM programming
- *Increased consistency and decoupled code*

Limitations:

- Currently static approach: No partial programming request, and within system '*version control*' is missing
- Complex derivations can still pose a problem to the AI, and understanding it's code and updating it might take more time than just writing it yourself
- Low context for AI: SAP etc. are not provided to the AI and it cannot assess specification correctness
- Currently more useful for validation site. Open question how to use this approach for both, production and validation

Future Directions:

- Better LLMs: GPT-5 was a significant leap for in SAS coding, but the future will only hold even better models.
- Bidirectional interactive session with multi file interpretation / checking / updating
- Advanced agent orchestration: Only one simply programming agent was used. Advanced systems (like the GitHub copilot) use multiple agents interacting with one another, performing tasks almost autonomously.
- Transferring the approach to TFL programming, offering much more time saving

The described approach comes with the necessity standardization

The following heuristics are strongly recommended to avoid the pitfall of standardizing

- **Bias toward defaults, not mandates.** → Provide recommended defaults that cover 80% of cases but always allow for opt-outs where possible.
- **Flexibility at the edges, stability at the core.** → Core principles (program structure) rarely change; peripheral details can evolve (section indicators)
- **Standardize the interfaces, not the implementations.** → Define how components interact but let teams optimize internals.

- Modern AI agent solutions offer a simple and efficient way of automating the statistical programming task of writing entire ADaM programs from specification.
- It is likely that In future, advanced statistical programming agents will act as proficient copilots, handling the vast majority of programming tasks
- The ultimate step in statistical programming will still rely with the programmer, but the way we work will change

And that's: **Why I can't see myself coding that much anymore.**

Thank you for Listening!

Contact me:

angelo.wibbeler@daiichisankyo.com