Paper SI01

Automating SDTM and ADaM Creation in Clinical Trials

Lyubov Sushchenko, Sanofi, Bridgewater, USA
Rie Ichihashi, Sanofi, Tokyo, Japan

ABSTRACT

Automation of SDTM and ADaM in clinical trials is a popular topic in the pharmaceutical industry nowadays. Many companies have a well-established standards team, which is defining CDISC and regulatory authority compliant standards within their company. Naturally, standardization of clinical data leads to automation initiatives.

In this paper, we will share Sanofi's innovative effort in standardizing the derivation or transformation specifications for data creation, with the intent to facilitate machine automation. We will introduce pseudo-codes, as a simple yet standard structured way of writing specifications, so that the programmer (either person or a machine) can understand and interpret the specifications consistently. This paper will provide examples of pseudo-code to demonstrate how it creates clear, specific, consistent, unambiguous derivation rules to enable automation. The end-to-end flow and the level of automation will also be discussed.

Defining and developing the standard specification is essential for automation to improve the quality of generated data, reduce programming time, and simplify mundane redundant tasks.
Keywords: standards, automation, pseudo-code, specification, derivation, transformation, ADaM

INTRODUCTION

In this paper, we will introduce to you the standard derivations in pseudo-code form. This will illustrate the automation of program creation, based on the pseudo-codes. We will outline the main concepts, give examples, and cover the process of program creation, using a code generator. Although we will focus mainly on ADaM examples, similar derivation principles are also applied to SDTM.

IMPORTANCE OF THE SPECIFICATIONS

Specifications are important to any programming. Clinical data programming is no exception. During study conduct, the necessity of good quality programming specification would often be emphasized.

What makes it so important? Clear, specific, consistent, unambiguous derivation rules are desirable at the beginning of the programming. It ensures the quality of the program as well as the efficiency of the programming. It also promotes collaboration among programmers. In essence, derivation rules are the language in which programmers communicate with each other, with statisticians, and with their data. Speaking the same language is critical. The information needs to flow consistently from protocol and SAP to the derivation rules, and translate into programs, no matter who does the programming (i.e., human or machine). Consequently, standardization of the derivation specification is the key to achieving the consistency and efficiency of the information flow.

ADAM AND THE CONCEPT OF THE PSEUDO-CODE

This paper will introduce a specific way of writing the specifications. For the automation project, we categorized different types of derivations used within our company for statistical analysis, which fell into 9 groups, we will discuss them in more detail in the section "Nine Standard Derivation Types". For some groups, we provided a standard pseudo-code and designed the schemas to collect the key elements of each pseudo-code for the derivation specification.

Pseudo-code is a simple yet standard structured way of writing specifications so that the programmer (either human or machine) can understand and interpret consistently. Conveniently, for each ADaM variable in ADaM metadata, a pseudo-code is created with standard derivation rules. See **Figure 1** below for an example where in ADaM Variable Metadata, the Pseudo Code column is highlighted in green. Pseudo-code is also applicable to value level derivations. See **Figure 2**.

Figure 1. Variable Metadata

Dataset Name	Relative Order	Variable Name	Variable Label	Type	Controlled Terminology	Source	Derivation	Assigned	Pseudo Code	Core	Instructions for Programmers
ADSL	170	AAGEU	Analysis Age Units	text	AGEU			Assigned as "[YEARS, MONTHS,	assigned	Cond	Conditionally required if AAGE is in
ADSL	180	AGEGR1	Pooled Age Group 1	text	[AGEGR1]		For AGE of [xxx] to [xxx], AGEGR1 = [xxx]		recode	Cond	Define AGE ranges in accordance if AGE=. then AGEGR1=""
ADSL	190	AGEGR2	Pooled Age Group 2	text	AGEGRPE		For AGE of [xxx] to [xxx], AGEGR2 = [xxx]		condderivation	Cond	AGEGR2 is reserved for EudraCT Preterm newborn - gestational age
ADSL	210	SEX	Sex	text	SEX	DM.SEX			source	Req	
ADSL	220	RACE	Race	text	RACE	DM.RACE			source	Req	If the SDTM RACE is "MULTIPLE",
ADSL	230	RACEOR	Original Race	text	RACEC	SUPPDM.RACEOR			source	Perm	
ADSL	240	RACEGRy	Pooled Race Group y	text	[RACEGRy]		For RACE of [xxx] or [xxx], RACEGRy = [xxx]		recode	Perm	Character description of a grouping used for special case when study h

Figure 2. WhereClauses, the main purpose is to provide metadata for VLM

Dataset Name	Variable Name	Where Variable	Comparator	Check Value	Value Label	Derivation	Pseudo Code
ADVS	AVAL	PARAMCD	EQ	WEIGHT	Weight (kg)	VS.VSSTRESN Where PARAMCD=VS.VSTESTCD	copy_stres
ADVS	AVAL	PARAMCD	EQ	DIABP	Diastolic Blood Pressure (mmHg)	VS.VSSTRESN Where PARAMCD=VS.VSTESTCD. An averag record is creted for triplicate measurements on the same date.	copy_stres, std_function(dtype_average)
ADVS	AVAL	PARAMCD	EQ	SYSBP	Systolic Blood Pressure (mmHg)	VS.VSSTRESN Where PARAMCD=VS.VSTESTCD. An averag record is creted for triplicate measurements on the same date.	copy_stres, std_function(dtype_average)

NINE STANDARD DERIVATION TYPES

In this section, we will give a brief introduction to nine standard derivation types with corresponding standard pseudo-codes.

We selected 1 to 3 datasets from each of the three standard CDISC ADaM data structures (ADSL, BDS, and OCCDS), investigated the derivation algorithms of each variable, and identified a total of 9 typical derivations. For each derivation type, we defined a pseudo-code.

This table is describing nine standard pseudo-codes, in the order from simple to complex derivation rules.

Pseudo Code	Description
source	Indicates the variable has an origin of “Predecessor” where the source variable is defined in the “Source” column. This is used mainly for SDTM predecessor variables
assigned	Assignment of a constant value for all the records in a dataset
derivation	Indicates that detailed derivation specs are available in the “Derivation” column in the “Variable Metadata” tab
recode	Indicates that a detailed derivation algorithm is available in the “recode” tab
rename	Renaming of a variable. This is used when a temporary variable is renamed into the ADaM dataset variable. The most common example is –DT, --DTM, and –TM variables. The ISO8601 datetime variables (e.g. EXSTDTC) in a source SDTM variable that is converted to temporary SAS date, datetime, and time variables (e.g. EXSTDT, EXSTDTM, and EXSTTM) early in the ADaM program, and renamed to suitable ADaM variable names (e.g. ADEX.ASTDT, ADSL.TRTSTDTM, etc.).
condderivation	Indicates that a detailed derivation algorithm is available in the “condderivation” tab
lookup*	Simple proc sql merge of 2 datasets, using subset condition if needed
std_function*	Complex derivations which involve multiple datasets and/or multiple steps of computations that cannot be classified into the derivation types described above
whereClauses*	Indicates that pseudo-code for value level parameters is populated in the VLM table in the “WhereClauses” tab

* Indicates pseudo-codes not presented in this paper

Currently, a user interface is under development, to help with entering study-specific values and interactively display the generated code. However, in this paper, we will use Excel to illustrate the writing of the pseudo-code. When the derivation rules are simple, pseudo-codes can be created along with the variable metadata.
For example, **Figure 3**, shows 3 simple pseudo-codes "source", "derivation", and "assigned". These pseudo-codes use the 3 corresponding variable metadata elements directly. Thus, the column which has the same name as the pseudo-code is used to pass the key element to the code generator.

Figure 3. Columns for additional information

Dataset Name	Variable Name	Variable Label	Type	Source	Derivation	Assigned	Pseudo Code
ADSL	STUDYID	Study Identifier	text	DM.STUDYID			source
ADSL	TRTDURD	Total Treatment Duration (Days)	integer		TRTEDT - TRTSDT + 1		derivation
ADEG	AWU	Analysis Window Unit	text			Assigned as "DAYS"	assigned

On the other hand, when the pseudo-code is "recode", "rename", "condderivation", "lookup", "std_function" or "whereclauses", additional information needs to be entered in another tab that has the same name as the pseudo-code.

Figure 4. Tabs for additional information

Dataset Name	Variable Name	Variable Label	Type	Source	Derivation	Assigned	Pseudo Code	Core	Instructions for Programmers
ADAE	AREL	Analysis Causality	text		AE.AEREL. If AEREL is missing, AREL=[xxx].		condderivation	Perm	For Pharma & Pasteur: If AEREL is derived due "Related" as per Safety Guidelines), AREL shou
ADAE	ARELN	Analysis Causality (N)	integer			Numeric code for AREL	recode	Perm	The numeric code for AREL. One-to-one map t

Dataset Metadata | Variable Metadata | recode | rename | condderivation | lookup | std_function | WhereClause

LET US DISCUSS NINE DERIVATION TYPES IN MORE DETAIL

1. Copy from a variable in the main source dataset (pseudo-code= "source"). See **Figure 5**.

This is used mainly for predecessor variables from SDTM. Information is parsed into program code to keep the variable from the main source dataset and used as a source for define.xml to display the predecessor's variable name. The main source dataset is the dataset that includes most of the source variables for that ADaM dataset (e.g., SDTM DM for ADSL).

Examples: ADSL.STUDYID, ADSL.USUBJID, ADSL.AGE, etc.

Figure 5. Pseudo-code "source" in the Variable Metadata tab

Dataset Name	Variable Name	Variable Label	Type	Source	Derivation	Assigned	Pseudo Code
ADSL	STUDYID	Study Identifier	text	DM.STUDYID			source
ADSL	USUBJID	Unique Subject Identifier	text	DM.USUBJID			source
ADSL	AGE	Age	integer	DM.AGE			source

... | Variable Metadata | recode | rename | condderivation | lookup | std_function | ...

The rendered SAS ® code is as follows:

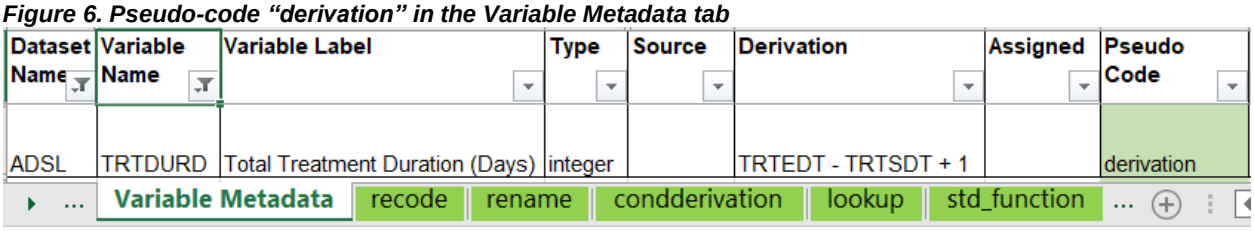
```
Keep STUDYID USUBJID AGE;
```

2. Simple computation which appears in define.xml as a computational algorithm is directly used as the program code (pseudo-code = "derivation"). See **Figure 6**.

For example, four basic arithmetic operations: addition, subtraction, multiplication, and division. The source variable(s) value(s) should be available in the same record in the main source dataset.

Example: Total Treatment Duration (Days) (ADSL.TRTDURD).

Figure 6. Pseudo-code "derivation" in the Variable Metadata tab



Dataset Name	Variable Name	Variable Label	Type	Source	Derivation	Assigned	Pseudo Code
ADSL	TRTDURD	Total Treatment Duration (Days)	integer		TRTEDT - TRTSDT + 1		derivation

... Variable Metadata | recode | rename | condderivation | lookup | std_function ...

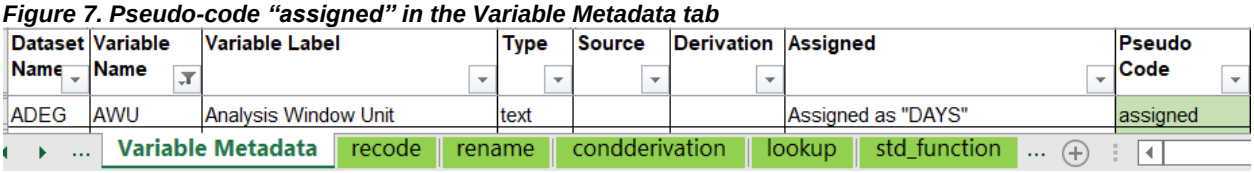
The rendered SAS ® code is as follows:

```
TRTDURD = TRTEDT – TRTSDT + 1;
```

3. Assignment of a constant value for all the records in a dataset (pseudo-code = "assigned"). See **Figure 7**. This is used when a variable has the same value for all the records in a dataset. The constant value needs to be entered into Variable Metadata. This information is parsed into program code to assign the constant value to the variable and used as a source for define.xml to display a comment for the variable as such: Assigned as "xxx".

Example: Analysis Window Unit (ADEG.AWU).

Figure 7. Pseudo-code "assigned" in the Variable Metadata tab



Dataset Name	Variable Name	Variable Label	Type	Source	Derivation	Assigned	Pseudo Code
ADEG	AWU	Analysis Window Unit	text			Assigned as "DAYS"	assigned

... Variable Metadata | recode | rename | condderivation | lookup | std_function ...

The rendered SAS ® code is as follows:

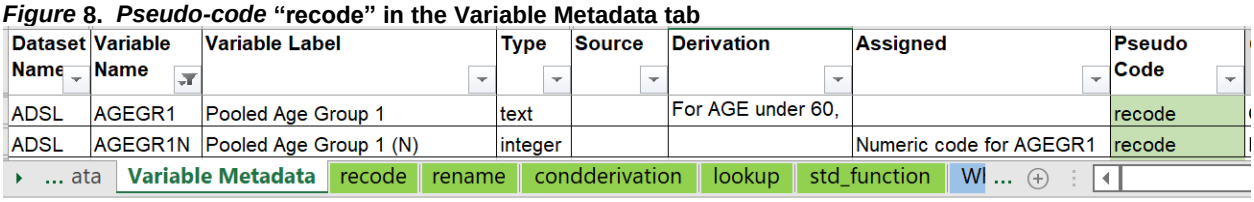
```
AWU = "DAYS";
```

4. Recode each value of a source variable to another value (pseudo-code = "recode"). See **Figure 8** for Variable Metadata conventions, and **Figure 9** for recode tab.

This is a convenient way for creating grouping variables, or numeric/character representations for existing variables. Users can enter multiple variables in the metadata as picture below. This convention could be used to derive multiple variables at the same time. In this example, we are deriving 2 related variables.

Example: Pooled Age Group 1 (ADSL.AGEGR1) and Pooled Age Group 1 (N) (ADSL.AGEGR1N)

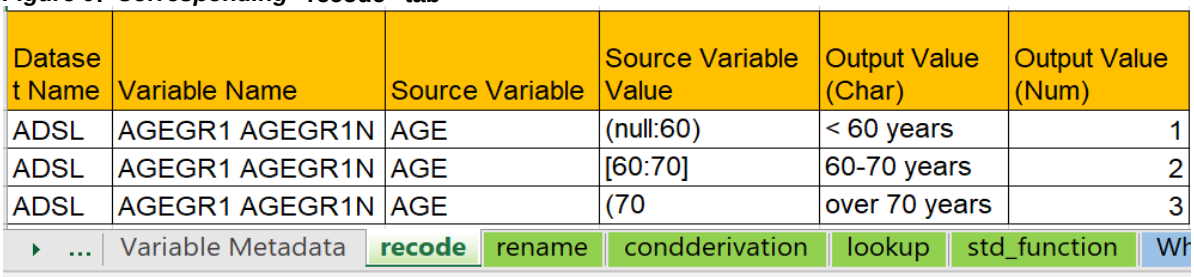
***Figure 8. Pseudo-code* "recode" in the Variable Metadata tab**



Dataset Name	Variable Name	Variable Label	Type	Source	Derivation	Assigned	Pseudo Code
ADSL	AGEGR1	Pooled Age Group 1	text		For AGE under 60,		recode
ADSL	AGEGR1N	Pooled Age Group 1 (N)	integer			Numeric code for AGEGR1	recode

... ata | Variable Metadata | recode | rename | condderivation | lookup | std_function | Wl ...

***Figure 9. Corresponding* "recode" tab**



Dataset Name	Variable Name	Source Variable	Source Variable Value	Output Value (Char)	Output Value (Num)
ADSL	AGEGR1 AGEGR1N	AGE	(null:60)	< 60 years	1
ADSL	AGEGR1 AGEGR1N	AGE	[60:70]	60-70 years	2
ADSL	AGEGR1 AGEGR1N	AGE	(70	over 70 years	3

... | Variable Metadata | recode | rename | condderivation | lookup | std_function | Wh

The rendered SAS ® code is as follows:

```
if . < AGE < 60 then do;
   AGEGR1 = "< 60 years ";
   AGEGR1N = 1;
end;
else if 60 <= AGE <= 70 then do;
   AGEGR1 = "60-70 years";
   AGEGR1N = 2;
end;
else if 70 < AGE then do;
   AGEGR1 = "over 70 years;
   AGEGR1N = 3;
end;
```

5. Rename a variable (pseudo-code = “rename”). See **Figure 10** for Variable Metadata conventions, and **Figure 11** for rename tab.

This is used when a temporary variable is renamed in the ADaM dataset. The most common example is --DT, --DTM, and --TM variables. The ISO8601 datetime variables (e.g., EXSTDTC) in a source SDTM variable that is converted to temporary numeric SAS date, datetime, and time variables (e.g., EXSTDT, EXSTDTM, and EXSTTM) early in the ADaM program. These temporary variables are renamed to suitable ADaM variable names (e.g., ADEX.ASTDT, ADSL.TRTSTDTM, etc.). The creation of temporary variables is streamlining program creation, when source data are being selected in ADaM programs, it is being processed generically, to ease user pseudo-code entry. And subsequentially helping with running the SAS program and deriving necessary variables without errors.

Example: Analysis Start Date (ADEX.ASTDT), Analysis Start Datetime (ADEX.ASTDTM), Analysis Start Time (ADEX.ASTTM)

Figure** 10. **Pseudo-code “rename” in the Variable Metadata tab

Dataset Name	Variable Name	Variable Label	Type	Source	Derivation	Assigned	Pseudo Code
ADEX	ASTDTM	Analysis Start Datetime	integer		or EX.EXSTDTC		dataset_derivation(DTC), rename
ADEX	ASTDT	Analysis Start Date	integer		date portion of		dataset_derivation(DTC), rename
ADEX	ASTTM	Analysis Start Time	integer		time portion of		dataset_derivation(DTC), rename

... Variable Metadata | recode | rename | condderivation | lookup | std_function | Where(...

Figure** 11. **Corresponding “rename” tab

Dataset Name	Variable Name	Source Variable Name
ADEX	ASTDTM ASTDT ASTTM	EXSTDTM EXSTDT EXSTTM

... | Variable Metadata | recode | rename | condderivation

The rendered SAS ® code is as follows:

```
rename EXSTDT = ASTDT
       EXSTDTM = ASTDTM
       EXSTTM = ASTTM;
```

6. Derivation of a variable based on if-then conditions of the dependent variable(s) (pseudo-code = “condderivation”). See **Figure 12** for Variable Metadata conventions, and **Figure 13** for condderivation tab.

The values can be entered as assigned pre-specified values or source variables or using a formula. The dependent variable(s) and source variable(s) should be available in the same record in the main source dataset.

Example: Analysis Relative Day (ADY). Derivations are different depends whether ADT is before TRTSDT or on/after TRTSDT

Figure 12. Pseudo-code "condderivation" in the Variable Metadata tab

Dataset Name	Variable Name	Variable Label	Type	Source	Derivation	Assigned	Pseudo Code
ADEX	ADY	Analysis Relative Day	integer		[ADT-TRTSDT+1 if ADT is on or after the treatment start date, ADT-TRTSDT if ADT precedes the treatment start date]		condderivation

... | Variable Metadata | recode | rename | condderivation | lookup | std_function | Where(...

Figure 13. Corresponding "condderivation" tab

Dataset Name	Variable Name	Derivation Condition	Output Variable Value	Source Variable or Formula
ADEX	ADY	. < ADT < TRTSDT		ADT - TRTSDT
ADEX	ADY	ADT >= TRTSDT > .		ADT - TRTSDT + 1

Dataset Metadata | Variable Metadata | recode | rename | condderivation | lookup | std_function | W

The rendered SAS ® code is as follows:

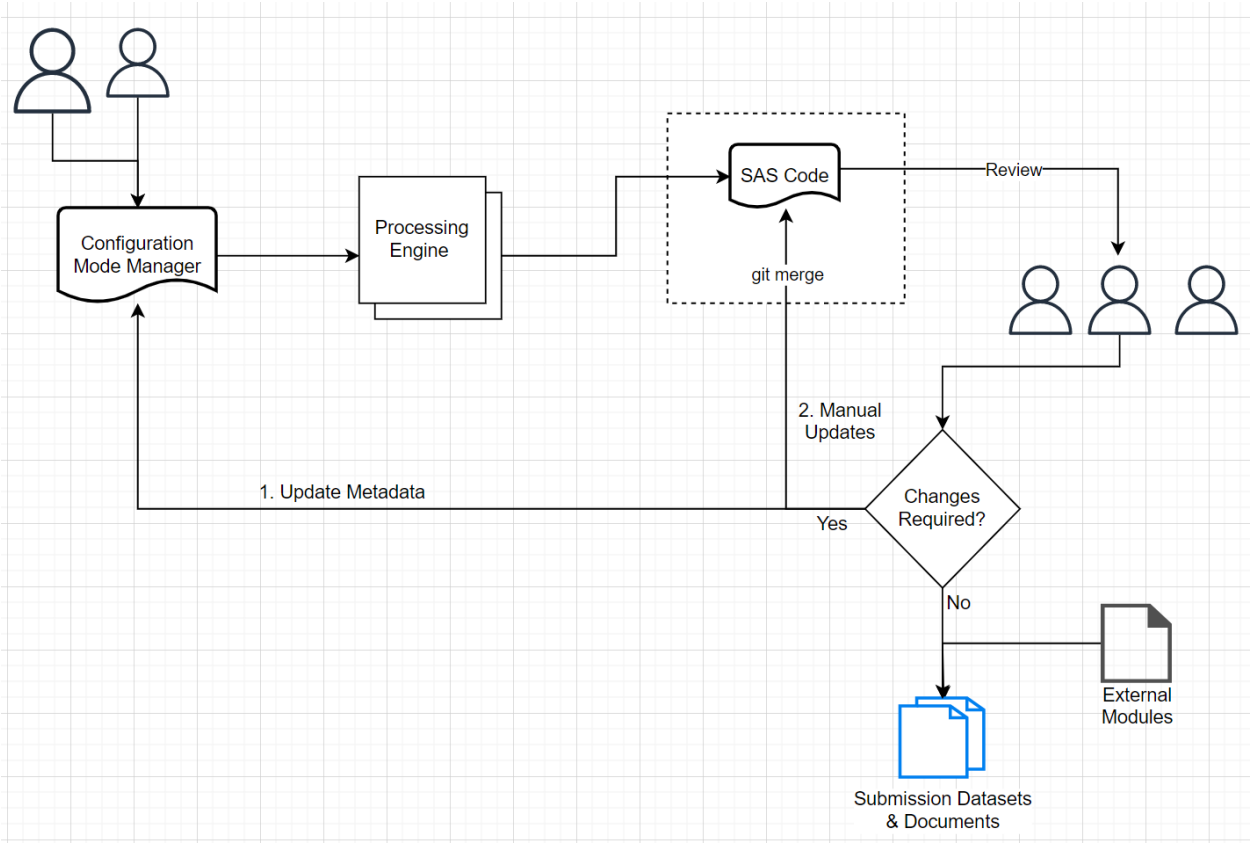
```
if   . < ADT < TRTSDT then ADY = ADT - TRTSDT;
  else if ADT >= TRTSDT >. then ADY = ADT - TRTSDT + 1;
```

Pseudo-codes for each variable can be defined at the global standard level, and users can adjust pseudo-code depending on the specifics of the study.

AUTOMATION OF PROGRAMS

To achieve automation, we are taking a configuration-based approach, generating the final programs for SDTM and ADaM. Leveraging the standard metadata, we are creating metadata transformation libraries used by the code generator. This configuration is fed into the code generator engine, which renders the final programs.
To derive a variable, we created a generic configuration model, that captures all necessary information. This model itself could be changed by the users, according to the complexity of the requirements. This creates a flexible model that is driven by the users and not a software developer.
The code generator is utilizing this configuration along with other parameters (program formats etc.) to generate output code. The users can choose in which language the output program is created. So far, we tested SAS and R.
A code generator is a software tool that automatically generates code based on input specifications or parameters. The generated code is intended to provide a starting point for a programmer, reducing the amount of manual coding required for a project. Code generators can generate code for a variety of purposes, such as creating database access layer code, generating user interface code, or generating code for specific design patterns. The generated code can be customized and refined to fit the needs of the specific project. The use of code generators can improve software development efficiency and reduce the likelihood of coding errors. **Figure 14** is describing the process flow.

Figure 14: **Process flow of program automation**



BENEFITS AND CHALLENGES OF PSEUDO-CODE

Even with the project still in development, these are the proposed benefits of the pseudo-codes:

- Agnostic
  Pseudo-code is language agnostic. As mentioned in the section “Automation of programs”, pseudo-code itself is not just programming code but a machine-readable derivation algorithm that can be translated into any programming code. The schema for each pseudo-code can be reused by any programming language. We have developed the template for each pseudo-code in SAS, and plan to develop another set of templates in R.

- Consistent
  Pseudo-code and additional user input in corresponding columns and tabs serve as the source for the code generator to generate the program as well as the derivation specifications for the programmers. Furthermore, for the three pseudo-codes: “source”, “derivation”, and “assigned”, the entries provided by the user “Source”, “Derivation”, and “Assigned” columns serve as a source for define.xml. We are developing a way to render the remaining 6 pseudo codes into define.xml attributes. This ultimately, will prevent programmers from entering the same information multiple times in specification documents, program code, and define.xml, and ensure consistency among them.

- Structured
  Pseudo-code provides a structure to the definition of derivation algorithms. When researching variable derivation algorithms for defining pseudo-codes, we realized, that current derivation specifications are not always evident for writing a program code. The level of clarity and detail varied from variable to variable and from study to study. And algorithms defined in the specs were different from author to author, from specification to specification. As a result, the same specification could be interpreted in multiple ways. Unlike specifications written in natural language, pseudo-code provides a concrete structured algorithm, requiring some user input, but in a pre-defined format with the same level of detail, and is interpreted uniquely.

We tested out code generator on several pilot studies, these were the challenges we faced:

- Pseudo-code and code generator functions are new concepts and need user training and support.
- Could be challenging at times to define correct/complicated pseudo-codes at the study level.
- This process adds a non-trivial layer of metadata which requires modern tools to be able to preview future SAS codes and enter the pseudo-codes.

CONCLUSION

In this paper, we introduced one company’s approach to the automation of dataset program creation. While this project is still under development, we are confident this approach will make routine everyday programming of clinical trial data easier and more efficient.

ACKNOWLEDGMENTS

The authors of this paper would like to express their gratitude to the Sanofi ART team, for their support and ideas, especially Faisal Mohammed. ADaM MAP team for their dedication and effort. B&P programmers for providing feedback and volunteering in testing out new tools. And Beilei Xu for inspiring to write this paper.

CONTACT INFORMATION

Your comments and questions are valued and encouraged.

Contact the authors at:

Author Name: Lyubov Sushchenko
Company: Sanofi US
Address: 55 Corporate Drive
City / Postcode: Bridgewater, 08807
Email: Lyubov.Sushchenko@sanofi.com
Web: https://www.sanofi.com

Author Name: Rie Ichihashi
Company: Sanofi K.K.
Address: Tokyo Opera City Tower, 3-20-3, Nishi Shinjuku
City / Postcode: Tokyo 163-1488, Japan
Email: Rie.Ichihashi@sanofi.com
Web: https://www.sanofi.com